A Novel Approach to Evolving Legacy Software Systems into a Grid Computing Environment

Ph.D. Thesis

Jianzhi Li

Software Technology Research Laboratory

De Montfort University

2006

Acknowledgement

I would like to express my gratitude to the many people who helped me in different ways with the development of this thesis. Without their continuous support and guidance, the completion of my research leading to this thesis would be impossible.

I wish to express my most sincere thanks to my supervisor Professor Hongji Yang for his invaluable advice, support and encouragement. I will carry out his guidance though out my life.

Meanwhile, I would like to thank colleagues at the Software Technology Research Laboratory at De Montfort University and for their support and feedback, and providing such a stimulating and friendly working atmosphere. It is fortunate for me to work and study with them.

This thesis would not been possible without the continuous support and guidance of my father Xiaodong, my mother Lifang and all my friends who gave me the strength and will to succeed.

Declaration

I declare that the work described in this thesis was originally carried out by me during the period of registration for the degree of Doctor of Philosophy at De Montfort University. from September 2003 to September 2006. Apart from the degree that this thesis is currently applying for, no other academic degree or award was applied by me based on this work.

Abstract

Grid computing is a new technology for the intent of sharing distributed resources and coordinated problem solving. On the other hand, legacy software systems can not be simply discarded as they are critical to business they support and because they encapsulate a great deal of knowledge and expertise about the applications. This research proposes an approach for evolving legacy software systems into Grid environment. The aim of this approach is to use legacy systems into Grid environment which enables the integration of legacy resources with Grid across distributed, dynamic environment and communities.

The methodology consists of multiple phases, include: using reverse engineering techniques to comprehend and decompose legacy system, employing AST, DTD and XSLT to transform and represent legacy system by XML as Grid components, and integrating these Grid components into Grid service environments. Also, the proposed approach is extended to the semantic Grid environment to carry out the initial step of the semantic Grid oriented legacy system evolution. As last, a legacy bank system case study is given. The purpose of this case study is to demonstrate that the proposed approach has the ability to evolve legacy systems into Grid service environments.

Different from related work, the approach proposed in this thesis provide an unified framework for Grid oriented legacy software system evolution. Through this research experience, it is argued that the detailed component mining approach needs to be tailored according to the features of a particular legacy system, and the legacy system evolution can assist Grid application development. The proposed approach is powerful for utilising reusable legacy resources into Grid environment to build Grid applications across distributed, dynamic environment and service oriented architecture communities.

Contents

Acknowledgement I
Declaration II
Abstract III
Contents IV
Table of Figures VIII
Chapter 1 1
Introduction 1
1.1. Proposed Research and Overview of Problem 1
1.2. Thesis Objectives 3
1.3. Contributions 5
1.4. Research Methods 6
1.5. Criteria for Success 7
1.6. Thesis Organisation 8
Chapter 2 10
Background 10
2.1. Grid Computing Technology 10
2.1.1. Grid Computing 10
2.1.2. Grid Services 12
2.1.3. Semantic Grid 13
2.2. Grid Technology Standards 14
2.2.1. Open Grid Service Architecture 15
2.2.2. Web Service Resource Framework 16
2.2.3. Key Components in Grid 18
2.3. Existing Grid Applications 19
2.3.1. Access Grid 19
2.3.2. SETI 20
2.3.3. MicroGrid 21
2.3.4. E-Science and ICENI 21
2.4. Future of Grid Technology 23
2.4.1. Visualisation 23
2.4.2. Virtual Organisation 24
2.4.3. Mobile/Wireless Grid 24
2.5. Other Distributed Object Technology 25
2.5.1. CORBA 26
2.5.2. J2EE 26
2.5.3. EJB 27
2.5.4. RMI 27

2.5.5. DCOM 28
2.5.6. Peer to Peer 29
2.5.7. Jini 30
2.5.8. Web Services 31
Chapter 3 32
Related Research 32
3.1. Software Evolution 32
3.1.1. Legacy System 32
3.1.2. Software Evolution and Reengineering 34
3.1.3. Software Maintenance 36
3.1.4. Software Restructuring 37
3.2. Software Reengineering 38
3.2.1. Component-Based Software Engineering 40
3.2.2. Software Clustering 41
3.2.3. Program Slicing 41
3.2.4. Wrapping Technique 42
3.3. Software Reuse for Grid 45
3.3.1. Grid Middleware Oriented Migration 45
3.3.2. Web Based System and Service Oriented System 47
3.3.3. Toward Grid Services 48
3.4. Summary 50
Chapter 4 51
Proposed Framework 51
4.1. Grid Oriented Evolution 51
4.2. Evolution Process 53
4.2.1. Component Identification 54
4.2.2. Grid Component Migration and Packing 56
4.2.3. Grid Service Integration 57
4.3. Summary 60
Chapter 5 62
Component Identification for Use in Grid Environment 62
5.1. Component in Grid Oriented Evolution 63
5.2. Definition of Legacy Grid Component 64
5.3. Design 66
5.4. Domain Analysis 67
5.5. Legacy System Decomposition 68
5.5.1. Decomposition Strategies 68
5.5.2. Slicing Rules 69
5.5.3. Slicing Algorithms 71
5.6. Using Hierarchical Cluster to Create Hierarchical Decomposition of Legacy Systems 74
5.6.1. Similarity Between Entities 76

5.6.2. Agglomerative Algorithms 78
5.6.3. Apply Agglomerative Hierarchical Clustering Analysis 79
5.7. Retargeting for Migration 83
5.8. Summary 84
Chapter 6 86
Grid Component Migration and Packing 86
6.1. Grid Middleware 86
6.2. Process Transfer 88
6.3. XML for Information Exchange 91
6.4. Integration of Legacy Resources and Grid with XML 92
6.5. Component Representation Using XML 94
6.5.1. AST Representation 95
6.5.2. DTD Expression 98
6.5.3. XSL Transformation for System Evolution 101
6.5.4. An Example 103
6.6. Wrap as XML Component 108
6.7. Summary 110
Chapter 7 112
Grid Service Integration 112
7.1. From Web Service to Grid Services 113
7.2. An Architecture for Grid Services Integration 115
7.2.1. Stateful Resources 115
7.2.2. Rationale 116
7.3. Grid Service Description 117
7.3.1. Stateful Resource Description 117
7.3.2. Accessing Stateful Resource 120
7.3.3. Stateful Resource Notification 122
7.4. Services Implementation 123
7.5. Services Repository 124
7.6. Grid Service Registration 125
7.7. Grid Service Resource Localisation 126
7.8. Grid Service Deployment 126
7.8.1. Service Group 128
7.8.2. Lifetime Management 129
7.8.3. Base Faults 129
7.9. Summary 130
Chapter 8 132
Extension to Semantic Grid 132
8.1. Retargeting for Migration to Semantic Grid Platform 133
8.2. Describing Grid Resource Metadata 133
8.2.1. Metadata in Semantic Grid 133

8.2.2. Grid RDF Model 134
8.2.3. Implementation 135
8.3. Migration to Semantic Grid Framework 138
8.3.1. Class Description of Semantic Grid Resources 138
8.3.2. Property Description of Semantic Grid Resources 139
8.4. Summary 140
Chapter 9 142
Case Study 142
9.1. Introduction 142
9.2. Experimentation Software Toolkit 142
9.2.1. Eclipses 142
9.2.2. Globus Toolkit 4 143
9.2.3. Tomcat and Sysdeo Tomcat Launcher 144
9.3. Components Identification 145
9.3.1. Program Slicing Analysis 146
9.3.2. Hierarchical Cluster Analysis 150
9.4. Grid Component Migration 153
9.4.1. Legacy Asset Representation 153
9.4.2. XML Component 157
9.5. Grid Service Integration 160
9.5.1. Service Resource Description 162
9.5.2. Implement Service 165
9.5.3. Define Deployment Parameters 169
9.5.4. Service Deployment 171
9.5.5. Running Grid Service 175
9.6. Summary 177
Chapter 10 178
Conclusions 178
10.1. Comparison and Evaluation 179
10.2. Summary 181
10.3. Assessment of Success 183
10.4. Conclusions 186
10.5. Future Work 187
Reference 189
Appendix A 205
Source Code of Example Legacy Bank System 205
Appendix B 236
Code/Specification of Case Study 236
Appendix C 249
List of Publications by Author 249

Table of Figures

Figure 4.1. Framework of Grid Services Oriented Legacy Software Systems Evolution. 54
Figure 4.2. Legacy Resources and Grid Services Integration. 59
Figure 5.1. An Example of a Control Flow Graph. 73
Figure 5.2. An Example of a Dendrogram. 82
Figure 6.1. Framework of Process Transfer in Grid Environment. 90
Figure 6.2. An Example of an Abstract Syntax Tree. 98
Figure 6.3. Screen-Shot at a Page of MLE System. 104
Figure 6.4. DTD Representation for a Page of MLE. 105
Figure 6.5. XML Representation of MLE Web Page. 106
Figure 6.6. XSLT transformation of MLE 107
Figure 6.7. New XML Representation of MLE Web Page 108
Figure 7.1. Stateful Resource Properties of MLE. 119
Figure 7.2. Request Message Representation 121
Figure 7.3. Response Message Representation. 122
Figure 7.4. The WSDD of MLE Services. 125
Figure 7.5. Service Deployment Structure. 128
Figure 8.1. A Page of 2005IRI Conference Website. 136
Figure 8.2. RDF/XML Representation of The Conference Information. 136
Figure 8.3. RDF/XML Representation of The Program Chair Information. 137
Figure 8.4. Class Description of Semantic Grid Resources. 139
Figure 8.5. Property Description of Semantic Grid Resources. 140
Figure 9.1. System Skeleton of Legacy Bank System. 147
Figure 9.2. System Call Graph of Legacy Bank System. 147
Figure 9.3. Program Call Graph of Credit Card Subsystem. 148
Figure 9.4. Original Program Fragment of Loan Payment Subroutine. 149
Figure 9.5. Control Flow Graph of Loan Payment Subroutine. 149
Figure 9.6. Slice of Loan Payment Subroutine with Respect to Criterion (14, {payment}). 150
Figure 9.7. Cluster Analysis of Legacy Bank System. 151
Figure 9.8. Dendrogram Output of Legacy Bank System. 152
Figure 9.9. Abstract Syntax Tree Representation for "interests = interest / 100 / 12" 154
Figure 9.10. Abstract Syntax Tree Representation for Loan Payment Subroutine. 155
Figure 9.11. DTD Representation of C Expression Statement. 156
Figure 9.12. DTD Representation of Loan Payment Subroutine. 157
Figure 9.13. Wrapping Loan Payment Subroutine with JNI. 159
Figure 9.14. XML Component Represented Application. 159
Figure 9.15. XML Component Represented Repository. 160
Figure 9.16. WSDL Document of CalculatorService. 164
Figure 9.17. QNames.java Document of CalculatorService. 166

Figure 9.18. Request Message Representation..... 167
Figure 9.19. Response Message Representation. 167
Figure 9.20. Service Implementation Document of CalculatorService..... 169
Figure 9.21. JNDI Deployment Document of CalculatorService..... 169
Figure 9.22. WSDD Deployment Document of CalculatorService..... 171
Figure 9.23. Buildservice.xml Document of CalculatorService..... 172
Figure 9.24. Buildservice.properties Document of CalculatorService..... 173
Figure 9.25. Updated Package Explorer..... 174
Figure 9.26. Build Successful Message. 175
Figure 9.27. Services Running Output. 176
Figure 9.28. Services Terminated Output..... 176

Chapter 1

Introduction

1.1. Proposed Research and Overview of Problem

It is well known that in industry circles, most desktop machines only use 5% to 10% of their capacity, and most servers barely peak out at 20%. What they need is not more horsepower, but more efficient use of existing horsepower. They need a way to tie all of these idle machines together into a pool of potential labour, and provide secure and reliable access to manage those resources [105]. As these organisations vary frequently in their purpose, scope, size, duration, and structure. Particularly, the resource configurations of organisations have the potential to be changed as well. Grid is a brand new technology for the intent of sharing distributed resources and coordinated problem solving, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications and high-performance orientation.

Grid services have emerged by combining Web services and Grid computing to perform a seamless information processing system across distributed, dynamic virtual organisations that the user can access from any location.

Affected by the Grid service orientation trend, many existing non-service-oriented software systems will turn into legacy systems. For such legacy systems, software evolution, as an approach to transforming a legacy system to an evolvable one, is the only way to extend their operational lifetime and make them capable of accommodating changes. Such an evolution process relies on a sound understanding of the original system via not only the wisdom and experiences of software developers but various

reverse engineering techniques [114]. These legacy systems require service oriented reengineering, which can facilitate legacy system evolution in Grid service orientated computing environments.

Making existing applications run in a Grid environment will increase hardware utilisation and resource sharing. From economic aspects, a business is constantly re-organising, changing its boundaries and reconfiguring its activities. Service-oriented reengineering enables legacy systems to adapt to continuous changes in business logic and market requirements. From technical aspects, application integration towards Grid and service oriented architecture will become common in Grid service oriented environment. Compared with design a new system, this approach is less risky and highly transport, and it is massively reduce time and cost for the enterprise.

With years of efforts, Grid researchers have successfully developed Grid technologies including security solutions, resource management protocols, information query protocols, and data management services. Due to the ultimate goal of Grid Computing is to design an infrastructure which supports dynamic, cross organisational resource sharing, there is a need of solutions for efficient evolving legacy systems into in the Grid environment. However, as Grid is a quite new technique, there are currently only few active researches related to the Grid oriented legacy system evolution.

This thesis focuses on establishing a general framework and methodology to assist with the evolution of legacy systems into Grid environments. This research proposes a solution for migrating legacy systems using a Grid user interface to enable the dynamic activation and Grid resources discovery. This ability of the legacy system to use Grid services will allow the enterprise to take advantage of the broad commercial support provided by Grid technology. The approach gives a way for legacy systems run in Grid environment as Grid services, and it enable legacy system used as distributed system for supply more working ability.

1.2. Thesis Objectives

To address the above challenges, a new approach of Grid oriented legacy software system evolution is proposed. First, reverse engineering techniques are used for program comprehension and design recovery. Then the legacy software systems are decomposed into a hierarchy of subsystems by defining relationships between the entities of the underlying paradigm of the legacy system. The decomposition is driven by program slicing and clustering techniques. Next, Grid components are created by wrapping objects and defining the interface. Finally, Grid components are allocated to Grid services environment by specifying the requirements of the system and characteristics of the network as an integer programming model.

In particular, this thesis aims to addressing the following research issues in the area of software evolution towards Grid oriented platform and system integration.

- Legacy system decomposition and components identification. In this part, the program slicing technique is used to decompose system, understand program, eliminate dead code and make selected code segments function independently by component interface parameters determination and deep source code comprehension. The clustering analysis technique is used to group large mounts of entities in a dataset and capture reusable legacy code segments into clusters according to their relationship and similarity from legacy systems, and to create a hierarchical structure of these reusable legacy code segments.

- Grid components migration and packing. This part presents the approach to migrating and packing the extracted legacy assets as Grid components and how achieve it. It is based on XML representation and transformation. Once a software component has been extracted from a legacy system, or has been built as a new component, its interface can be extracted and represented in XML. The

representation is finished not only by the component wrap, but also with the sources code analysis, such as the using of AST, DTD and XSLT. At last, the legacy components are wrapped as XML Grid components which could be further used in the Grid services environment.

- Grid services integration. This part presents a framework that integrates the XML packaged legacy system components into the Grid service environment. It describes a method for defining the legacy resources as stateful resources, and building the Grid services based on these reusable resources. There are four major steps to migrate components in the new Grid services environment. In the first step, the services' properties and its interface are defined by WSDL. Then the implementation of the service is carried out by Java. In the third step, the WSDD and JNDI file define the deployment parameters including services registration and resources localisation. The last step is to deploy the Grid service. This Grid service oriented reengineering methodology brings more flexibility, expansibility and reusability, as well as more reliability.

- Extension to semantic Grid environment. This part proposed an approach to reusing the legacy system assets into the semantic Grid framework. This approach is based on source code translation and reconstruction, component reusing and the semantic Grid framework retargeting. XML is employed to provide a simple and elegant frame for describing the properties of the reusable legacy system resources as Grid RDF data models which are based on a specific XML mark language RDF/XML. The RDF/XML representation and RDF-schema description are key techniques in reusing recovered legacy components in semantic Grid framework.

1.3. Contributions

The contributions of this thesis include:

- A unified approach integrates the software evolution approach with the Grid technique. It utilises reusable legacy resources into Grid environment to build Grid applications across distributed, dynamic environment and service oriented architecture communities. This research develops an effective way to reuse legacy assets with Grid technology.

- Identify components from legacy software systems for the use in Grid environment with reverse engineering techniques such as program slicing and software clustering.

- Specify Grid XML components which are migrated from legacy system components and integrate them into the Grid service framework.

- A XML representation and transformation method that transforms between legacy files and XML documents. The representation is finished not only by the component wrap, but also with the sources code analysis included such as the using of AST, DTD and XSLT.

- Using the component based design approach into the Grid application development area.

- An evolvable Grid oriented framework can integrate the XML packaged legacy system components into the Grid service environment.

- An evolvable Grid oriented framework can reuse the legacy system assets into the semantic Grid environment.

1.4. Research Methods

There are a lot of research methods being used within the field of computing science. These include but are not limited to implementation driven research, mathematical proof techniques and empiricism [83]. The implementation driven research approach progresses by iteratively building better and better systems. But if the system fails then only few insights into the basic research question may be gained, although the failure due to the limitations of the implementation may be more often than the failure due to the idea itself. And this method may be difficult to generalise from a specific system to generic principles. The mathematical proof techniques approach uses formal proofs to reason about the validity of a hypothesis given some evidence. But the mathematical abstractions used in a proof can be too abstract or generic, so that they completely ignore critical issues that must be considered during the implementation of a particular system.

In this thesis, the research method relies on the empiricism which has been used to support many different aspects of research within computing science. The proposed research method in this thesis can be summarised in the following four stages: hypothesis, methods, results and conclusion.

In hypothesis stage, Chapter 4 presents the framework on Grid oriented evolution approach and evolution process of the framework. It aims to identify and extract components from legacy software systems, and to generate suitable Grid component for different Grid systems and integrate the evolved components in Grid environment. Such a reengineering framework is composed of legacy system decomposition, component identification, Grid components composition and Grid environment integration that are based on current feasible Grid technology. This chapter explicitly identifies the ideas that are to be tested by the research.

In method stage, Chapter 5 proposes an approach to identifying concerned resources from legacy systems for designing components which are used in Grid environment, including

the legacy system decomposition with program slicing and hierarchical decomposition creation with software clustering. Chapter 6 describes a Grid component migration and packing approach based on XML representation and transformation with the using of AST, DTD and XSLT. Chapter 7 presents a framework that integrates the XML packaged legacy system components into the Grid service environment, including Grid service resources description, services implementation, service registration, resources localisation and service deployment. Chapter 8 extends the proposed approach into the semantic Grid environment and takes the initial work of reusing the legacy system assets into the semantic Grid environment. These four chapters identify the techniques that will be used in order to establish the hypothesis, and explicitly describes how they are collaborated to achieve the hypothesis.

In the stage of results,. Chapter 9 presents a case study related to decompose legacy system, represent legacy resources as Grid component and integrate them to the Grid service framework. This chapter presents and compiles the results that have been gathered from the proposed method.

Finally in conclusion stage, Chapter 10 concludes the thesis and states the supported hypothesis. The possible future works are presented as well.

1.5. Criteria for Success

This thesis proposed a unified approach to evolving legacy software systems into Grid environment. It includes identify concerned resources from legacy systems based on reverse engineering techniques, represented legacy assets by XML and reuse these legacy resources in the Grid environment include the Grid services and semantic Grid framework.

The following criteria are given to judge the success of the research described in this

thesis:

- Can this approach handle the diversity of Grid based systems?
- Does the legacy system component identification and Grid XML component representation methods effective and feasible for the Grid oriented legacy software system evolution process?
- Does this approach improve the efficiency Grid development industry?
- Is the approach feasible for realisation? For example, is it possible to build a practical tool based on the approach?
- Is the approach capable for industrial-scaled systems?

1.6. Thesis Organisation

The rest of this thesis is organised as follows:

- Chapter 1 gives the research problem, motivation, scope, contribution and research methods of this thesis.
- Chapter 2 provides an overview of Grid technology, software evolution and it introduces some approaches and techniques which are used in the Grid oriented software evolution. It also answers such questions as what is Grid and what the Grid oriented software evolution requires.
- Chapter 3 gives an overview of the related researches in the area of Grid technologies, Grid system development and system evolution for Grid environment.

- Chapter 4 presents the framework on Grid oriented evolution approach and evolution process of the framework. It also presents the key components and challenges in the evolution process.

- Chapter 5 explains component identification approach for use in Grid environment based on software reverse engineering techniques such as program slicing and software clustering.

- Chapter 6 introduces the concept of Grid components migration and packing, and how they have been developed in the proposed approach.

- Chapter 7 discusses how the Grid components can be integrated into the Grid services framework.

- Chapter 8 explains the solutions of extending the previous approach for the legacy systems to meet the semantic Gird structure.

- Chapter 9 presents a case study to demonstrate the proposed approach of Grid services oriented legacy software system evolution.

- Chapter 10 concludes the thesis and makes a discussion on possible future research.

Chapter 2

Background

2.1. Grid Computing Technology

2.1.1. Grid Computing

Grid computing [18] is an emerging computing model that provides the ability to perform higher throughput computing by taking advantage of many networked computers to model a virtual computer architecture that is able to distribute process execution across a parallel infrastructure. Grid uses the resources of many separate computers connected by a network (usually the Internet) to solve large-scale computation problems [42]. Grid provides the ability to perform computations on large data sets, by breaking them down into many smaller ones, or providing the ability to perform more computations at once than a single computer would perform and by modelling a parallel division of labour between processes. Currently, there are may kinds of Grid exist, such as data Grid, resources Grid and information Grid.

Grid computing is distributed computing taken to the next evolutionary level [120]. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected systems sharing various combinations of resources.

Grid computing enables the virtualisation of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities. Just as an

Internet user views a unified instance of content via the Web, a Grid user essentially sees a single, large virtual computer.

Benefits of Grid Technology include:

- Increasing hardware utilisation and resource sharing.
- Enabling companies to scale out incrementally with low-cost components.
- Reducing management and administration requirements.
- Accelerate processing time to get the results.
- Enable collaboration and promote operational flexibility.
- Efficiently scale to meet variable business demands.
- Increase productivity.
- Leverage existing capital investments.
- Infrastructure optimisation.
- Increase access to data and collaboration.
- Resilient, highly available infrastructure.

Grid are composed of Virtual Organisations (VO) using a common suite of protocols. It supports the sharing and coordinated use of diverse resources in dynamic VOs [44]. Virtual organisations, as explained above, can be a handful of servers or desktop PCs in a single room, or a heterogeneous system scattered around the world connected via the Internet [45]. All of these systems are able to work together because of certain protocols, which control connectivity, resource allocation and management, and coordination of those resources.

Grid computing is an emerging technology, so different people will give different definitions. But almost all definitions are focused on large-scale resource sharing, innovative applications, and in some case, high-performance orientations.

The concept of utility computing is independent of any specific standard or technology. Its goal is to meet the needs of dynamic, distributed resource sharing across every application area and industry segment. Grid computing can be considered as a specialised instance of utility computing, driven up to now by the needs of the scientific community.

The Grid is not only a computing infrastructure, for large applications, it is also a technology that can bond and unify remote and diverse distributed resources ranging from meteorological sensors to data vaults and from parallel supercomputers to personal digital organisers [9].

2.1.2. Grid Services

Web services technology is a set of specifications for 'packaging technology' that allows functions of a software system to be published using Web Service Definition Language (WSDL), discovered using Universal Description, Discovery and Integration (UDDI), executed using Simple Object Access Protocol (SOAP) [81].

Grid services are emerged by integrating Grid computing and Web services to perform a seamless information processing system across distributed, heterogeneous, dynamic virtual organisations that the user can access from any location. It allows the entire collection to be seen as a seamless information processing system that the user can access from any location.

Web services are the technology of choice for Internet based applications with loosely coupled clients and servers. But the implementations of Web services are typically stateless. Compared with this, Grid services are evolved to make possible of dynamically sharing and coordinate dispersing of heterogeneous services resources.

Grid services are basically Web services with improved characteristics and services. The most important improvement is that Grid services are stateful and transient services. It can remember what have been done from one invocation to another with the support of stateful resources.

The definition of stateful resources [46] is given as resources which have a specific set of state data expressible as an XML document and a well-defined lifecycle, and be known to, and acted upon, by one or more Web services. Its state may be implemented as an actual XML document that is stored in memory, in the file system, in a database, or in some XML Repository.

2.1.3. Semantic Grid

The semantic Grid [119] refers to an approach to Grid computing in which information, computing resources and services are described in standard ways that can be processed by computer [60]. This makes it easier for resources to be discovered and joined up automatically, which helps bring resources together to create virtual organisations.

The semantic Grid is an extension of the current Grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation [136]. This notion of the semantic Grid was first articulated in the context of e-Science, observing that such an approach is necessary to achieve a high degree of easy-to-use and seamless automation, enabling flexible collaborations and computations on a global scale.

Semantic Grid is based on the Resource Description Framework (RDF) [65], which integrates a variety of applications using XML for syntax [95] and URIs for naming [118].

The RDF describes metadata [26] and provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning [55]. And it provides a framework that allows data to be shared and reused across application,

enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners.

The use of semantic Web and other knowledge technologies in Grid applications is sometimes described as the knowledge Grid. Semantic Grid extends this concept by also applying these technologies within the Grid middleware. Some semantic Grid activities are coordinated through the semantic Grid research group of the Global Grid Forum (GGF).

2.2. Grid Technology Standards

There must be standards for Grid computing that will allow a secure and robust infrastructure to be built. The main standardisation initiatives for the Grid come from the Global Grid Forum (GGF) and Organisation of the Advancement of Structured Information Standards (OASIS).

GGF is a forum initiated by a community of individuals from industry and research leading the global standardisation effort for the Grid. GGF's primary objectives are to promote and support the development, deployment, and implementation of Grid technologies and applications via the creation and documentation of best practices technical specifications, user experiences, and implementation guidelines.

OASIS is a not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. The consortium produces more Web services standards than any other organisation and is responsible for the development of the Web Services Resource Framework (WSRF) [47].

2.2.1. Open Grid Service Architecture

The initial defining architecture standard for the Grid is the Open Grid Services Architecture (OGSA) specification which provides the Open Grid Services Infrastructure (OGSI) [141] of the GGF in June 2002 [24]. OGSA defines a set of common interfaces for the important services in a Grid and also defines a common set of Grid services which all Grid should offer [33].

The Open Grid Services Infrastructure specification version 1.0, released in July 2003 [54], defines a set of conventions and extensions on the use of WSDL and XML Schema to enable stateful Web services [134]. The four main layers comprise in the OGSA architecture in is shown in [144]:

Physical and logical resources layer

The concept of resources is the centre to OGSA and to Grid computing. Resources comprise the capabilities of the Grid, and are not limited to processors. Physical resources include servers, storage, and network. Above the physical resources are logical resources. They provide additional function by virtualising and aggregating the resources in the physical layer. General purpose middleware such as file systems, database managers, directories, and workflow managers provide these abstract services on top of the physical Grid.

Web services layer

The second layer in the OGSA architecture is Web services. Here is an important tenet of OGSA: All Grid resources (both logical and physical) are modelled as services. OGSI specification defines Grid services and builds on top of standard Web services technology. OGSI exploits the mechanisms of Web services like XML and WSDL to specify standard interfaces, behaviours, and interaction for all Grid resources. OGSI extends the definition

of Web services to provide capabilities for dynamic, stateful, and manageable Web services that are required to model the resources of the Grid.

OGSA architected Grid services layer

The Web services layer, with its OGSI extensions, provides a base infrastructure for the next layer - architected Grid services. The Global Grid Forum is currently working on defining many of these architected Grid services in areas like program execution, data services, and core services. Some architected Grid services are already defined, and some implementations of these architected services have already appeared. As implementations of these newly architected services begin to appear, OGSA will become a more useful SOA.

Grid applications layer

Over time, as a rich set of Grid-architected services continues to be developed, new Grid applications that use one or more Grid architected services will appear. These applications comprise the fourth main layer of the OGSA architecture.

The OGSA supports the creation, maintenance, and application of ensembles of services maintained by VOs. It is certainly an important step in developing a service-oriented architecture for Grid. With the delivery of the initial implementations of OGSI, OGSA is getting ready to accelerate its entry into mainstream commercial computing environments [135].

2.2.2. Web Service Resource Framework

Web Service Resource Framework (WSRF), a set of proposed Web services specifications that define a rendering of the WS-Resource approach in terms of specific message exchanges and related XML definitions.

Because there are many inconvenient of OGSI, such as OGSA which has too much stuff in one specification, so it does not work well with existing Web services and XML tooling, it is too object oriented and OGSA do not support the forthcoming WSDL 2.0. The WSRF proposal is an evolution of OGSI concepts, it make Grid technology batter use with Web services [32].

The WS-Resource Framework is defined by following normative specifications:

- WS-Resource Properties. WS-Resource Properties [49] is the definition of a WS-Resource, and mechanisms for retrieving, changing, and deleting WS-Resource properties.
- WS-Notification. WS-Notification [53] defines a general, topic-based Web service system for publishing and subscribeing interactions that build on the WS-Resource framework.
- WS-Resource Lifetime. Mechanisms for WS-Resource destruction include message exchanges that allow a requestor to destroy a WS-Resource, either immediately or by using a time-based scheduled resource termination mechanism [51].
- WS-Base Faults. WS-Base Faults [52] is a base fault XML type for use when returning faults in a Web services message exchange.
- WS-Service Group. WS-Service Group [50] is an interface to heterogeneous by-reference collections of Web services.
- WS-Renewable References. A conventional decoration of a WS-Addressing [22] endpoint reference with policy information is needed to retrieve an updated version of an endpoint reference when it becomes invalid.

2.2.3. Key Components in Grid

Grid may consist of many high level key components. Depending on the Grid design and its expected use, some of these components may or may not be required, and in some cases they may be combined to form a hybrid component [80].

Portal/user interface. A Grid portal provides the interface for a user to launch applications that will use the resources and services provided by the Grid. From this perspective, the Grid seems as a virtual computing resource for users just as the receptacle seems as an interface to a virtual generator to consumer power.

Security. At the base of Grid environment, there must be mechanisms to provide security, including authentication, authorisation, data encryption, and so on. The Grid Security Infrastructure (GSI) component provided by Globus is an OpenSSL implementation. It also provides a single sign-on mechanism, so that once a user is authenticated, a proxy certificate is created and used when performing actions within the Grid.

Broker. Once authenticated, the user will launch an application. Based on the application, and possibly on other parameters provided by the user, the next step is to identify the available and appropriate resources to use within the Grid. This task could be carried out by a broker which provides information about the available resources within the Grid and their status [110].

Scheduler. Once the resources have been identified, the next logical step is to schedule the individual jobs to run on them. If a set of stand-alone jobs are to be executed with no interdependencies, then a specialised scheduler may not be required. A higher level scheduler might be used to schedule work to be done on a cluster, while the cluster's scheduler would handle the actual scheduling work on the cluster's individual nodes.

Data management. Data management is needed if any data must be moved or made accessible to the nodes where an application's jobs will execute. This component, know

as Grid Access to Secondary Storage, includes facilities such as GridFTP which is built on top of the standard FTP protocol, but adds additional functions and utilises the GSI for user authentication and authorisation. This facility provides third-party file transfer so that one node can initiate a file transfer between two other nodes.

Job and resource management. The Grid Resource Allocation Manager (GRAM) provides the services to actually launch a job on a particular resource, check its status, and retrieve its results when it is complete.

Other facilities. There are other facilities that may need to be included in Grid environment and considered when designing and implementing application. For instance, inter-process communication and accounting services are two common facilities that are often required.

2.3. Existing Grid Applications

2.3.1. Access Grid

Access Grid (AG) [30] is a collection of resources and technologies that enables large format audio and video based collaboration between groups of people in different locations. The Access Grid is an ensemble of resources, including multimedia large-format displays, presentation and interactive environments, and interfaces with Grid computing middleware and visualisation environments. In simple terms, it is advanced video conferencing using big displays and with multiple cameras at each node (site). The technology was invented at Argonne National Laboratory, Chicago.

The Access Grid is an ensemble of resources including multimedia large-format displays, presentation and interactive environments, and interfaces to Grid middleware and to visualisation environments. These resources are used to support group-to-group interactions across the Grid. For example, the Access Grid is used for large scale

distributed meetings, collaborative work sessions, seminars, lectures, tutorials, and training. The Access Grid thus differs from desktop-to-desktop tools that focus on individual communication.

The Access Grid has over 1,500 users worldwide. Each institution has one or more AG nodes, that contains the high-end audio and visual technology needed to provide a high quality compelling user experience. The nodes are also used as a research environment for the development of distributed data and visualisation corridors and the study of issues relating to collaborative work in distributed environments.

2.3.2. SETI

SETI@home [3] or the Search for Extraterrestrial Intelligence, is a Grid computing project using Internet-connected computers in the search for extraterrestrial intelligence, hosted by the Space Sciences Laboratory, at the University of California, Berkeley, in the United States. It is a scientific effort seeking to determine if there is intelligent life outside Earth.

Most of the SETI programs used today analyse the data from the telescope in real time. To tease out the all signals, a great amount of computer power is necessary. SETI programs could never afford to build or buy that computing power. Rather than using a huge amount of computer to do the job, SETI programs less computers but just take longer time to do it.

By using Grid technology, SETI team develop an approach to use computers all around the world. As long as authorised, SETI could use the computer power ability to help them analysing single data. They do this with a screen saver that can get a chunk of data from SETI over the internet, analyse these data, and then report the results back to them. When users need their computer back, the screen saver instantly gets out of the way and only continues the analysis when the computer is free.

SETI@home also has been used as a stress testing tool for computer workstations, as it runs the computer CPU at full power for a sustained time period. This is especially useful to over lockers. The results of the data processing are normally automatically transmitted when the computer is connected to the internet. It can also be instructed to connect to the internet as needed.

2.3.3. MicroGrid

The MicroGrid [129] provides online simulation of large-scale (20,000 router, thousands of resources) network and Grid resources. By creating a virtual Grid environment in which existing middleware and applications can be run unchanged, detailed study of complex dynamic behaviour such as scaling, failure responses, and other emergent behaviour that can be explored.

The MicroGrid complements experimentation with actual Grid by supporting exploration of a wide variety of Grid resource configurations and scenarios (such as catastrophic failure) are not be possible to exhibit in the actual resource. Other advantages of MicroGrid include the ability to explore a wide range of resource (network, compute, storage) environments, dynamic competitive loads, reduced experimental effort, and increased observability.

2.3.4. E-Science and ICENI

The term e-Science has been defined by John Taylor (Director General of the UK Research Councils) as "global collaboration in key areas of science and the next generation of infrastructure that will enable it." The Grid is an architecture proposed to bring all these issues together and make a reality of such a vision for e-Science. E-Science is used to describe computationally intensive science that is carried out in highly distributed network environments, or science that uses immense data sets that require Grid computing. In the

future, e-Science will refer to the large scale science that will increasingly be carried out through distributed global collaborations enabled by the Internet. Typically, a feature of such collaborative scientific enterprises is that they will require access to very large data collections, very large scale computing resources and high performance visualisation back to the individual user scientists.

Imperial College e-Science Network Infrastructure (ICENI) [56] is a typical Grid computing platform. It provides high-level abstractions for e-science which will allow users to construct and define their own applications through a graphical composition tool integrated with distributed component repositories and deliver this environment across a range of platforms and devices [57]. It is a Grid middleware system, which consists of a service oriented architecture and an augmented component programming model being developed at the London e-Science Centre. This platform independent framework explores effective application execution upon distributed federated resources through open and extensible XML derived protocols and a Java implementation.

ICENI framework uses a component programming model to describe Grid applications [97]. This is clearly a benefit because it promotes code reuse and reduces the task of Grid application development to that of application composition. The scheduling of component applications in this environment takes place through the ICENI scheduling framework which allows 'pluggable' scheduling algorithms and platform specific launchers.

Recovered legacy components communicate in XML-based messages with the help of XML parser. From user's aspect, the framework supply visualisation Grid services which enable consistent resource access across multiple heterogeneous platforms without regard for how these services are implemented. It enables mapping of multiple logical resource instances onto the same physical resource and facilitates management of resources within a Virtual Organisation based on composition from lower-level resources.

Furthermore, it also allows to compose basic services to form more sophisticated services and underpin the ability to map common service behaviour seamlessly onto native platform facilities. The distributed component repositories will allow developers to contribute their software to the wider community and easily incorporate other work into their own components or applications. The ICENI middleware federates the underlying resources that enable the e-scientist to carry out their work by allowing sophisticated and extensible access and usage policies to be specified.

2.4. Future of Grid Technology

Many features for future of Grid technology have been designed. These features essentially change the current ways of using software and services. They will supply and brand new platform for both enterprise and users. Also, they will provide more conveniences and practicalities.

2.4.1. Visualisation

Visualisation enables consistent resource access across multiple heterogeneous platforms. Virtualisation also enables mapping of multiple logical resource instances onto the same physical resource and facilitates management of resources within a virtual organisation based on composition from lower-level resources. Further, virtualisation can compose basic services to form more sophisticated services without regard for how these services are implemented.

Visualisation Grid [44] services also underpins the ability to map common service semantic behaviour seamlessly onto native platform facilities. This virtualisation is easier if the service functions can be expressed in a standard form, so that any implementation of a service is invoked in the same manner.

2.4.2. Virtual Organisation

A virtual organisation(VO) [43] is a collection of resource providers and consumers with a set of clearly and carefully defined sharing rules, such as what is shared, who is allowed to share, the conditions under which sharing occurs, etc.

The sharing which are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem solving and resource brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and institutions defined by such sharing rules form a virtual organisation.

2.4.3. Mobile/Wireless Grid

As the rapid growth of the wireless/mobile networks and wireless/mobile devices technologies, the amount of their uses and the improvement of performance go on increasing rapidly.

Mobile/Wireless Grid [112] enables mobile devices (mobile phone, PDA, laptop computer) to access native Grid anytime and anyplace. It helps mobile devices to use idle resources of a great number of other wireless devices as well as wired Grid resource. This will being great benefits to mobiles devices. For this instances, people can use a PDA to watch movie without add a hard disc or faster memory and CPU. One day, mobile devices can be used as a typical computer, or even a supercomputer without spending more money.

As there are much advantages of Grid technology, more and more applications are involved in Grid technology. Based on this idea, getting more applications from integration with Grid could be a tremendous work in the future.

2.5. Other Distributed Object Technology

Grid is an emerging computing model that provides the ability to perform higher throughput computing by taking advantage of many networked computers to model a virtual computer architecture that is able to distribute process execution across a parallel infrastructure.

Other enterprise development technologies such as DCOM, Enterprise Java Beans (EJB), Java 2 Enterprise Edition (J2EE), and CORBA are all systems designed to enable the construction of distributed applications. They provide standard resource interfaces, remote invocation mechanisms, and trading services for discovery and hence make it easy to share resources within a single organisation.

Comparatively, Grid use the resources of many separate computers connected by a network (usually the Internet) to solve large-scale computation problems. Grid provide the ability to perform computations on large data sets, by breaking them down into many smaller ones, or provide the ability to perform many more computations at once than would be possible on a single computer, by modelling a parallel division of labour between processes. Today resource allocation in a Grid is done in accordance with service level agreements. As a type of distributed computing, the Grid should be characterised by other related technologies.

The most widely accepted standard distributed object technologies include: Object Management Group's Common Object Request Broker Architecture (CORBA), Sun's Java remote method invocation (RMI), Sun's Enterprise JavaBeans (EJBs) and

Microsoft's Distributed Component Object Model (DCOM). Also, some other related technologies such as P2P, JINI and Web services will be reviewed.

2.5.1. CORBA

In computing, Common Object Request Broker Architecture (CORBA) [126] is a standard for software componentry, created and controlled by the Object Management Group (OMG). It defines APIs, communication protocol, and object/service information models to enable heterogeneous applications written in various languages running on various platforms to interoperate. CORBA therefore provides platform and location transparency for sharing well-defined objects across a distributed computing platform.

CORBA uses an interface definition language (IDL) to specify the interfaces that objects will present to the world. CORBA then specifies a "mapping" from IDL to a specific implementation language like C++ or Java [67]. This mapping precisely describes how the CORBA data types are to be used in both client and server implementations. Standard mappings exist for Ada, C, C++, Lisp, Smalltalk, Java, and Python [109].

CORBA is an integration technology, but not a programming technology. As such, it is used to connect distributed objects and integrate them with other heterogeneous computing environments.

2.5.2. J2EE

Java 2 Platform Enterprise Edition (J2EE) [2] technology provides a component-based approach to design, development, and deployment of Web enabled enterprise applications. J2EE offers a multi-tier architecture that separates presentation logic, business logic and back-end services or database. J2EE consists of different components to assist the fast development and deployment of each tier, including:

- Java Applications and applets running on client machines to access to Web servers.

• Java Server Page (JSP) [15] and Java Servlet [75] running on Web servers to receive and responses clients' request.

• Enterprise JavaBeans running on the application severs to implement business logic.

2.5.3. EJB

Enterprise JavaBeans (EJB) [102] technology is the server-side component architecture for J2EE. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology.

The EJB specification details how an application server provides persistence, transaction processing, concurrency control, events using Java message service, naming and directory services (JNDI) [62], security (JCE and JAAS), deployment of software components in an application server and remote procedure calls using RMI-IIOP or CORBA. It also defines the roles played by the EJB container and the EJBs as well as how to deploy the EJBs in a container.

EJBs are deployed in an EJB container within the application server. And each EJB must provide a Java implementation class and two Java interfaces. Because these are merely Java interfaces and non-concrete classes, the EJB container must generate classes for these interfaces that will act as a proxy in the client. Client code invokes a method on the generated proxies, which in turn places the method arguments into a message and sends the message to the EJB server. The proxies use RMI-IIOP to communicate with the EJB server.

2.5.4. RMI

The Java Remote Method Invocation [36] API, or Java RMI, is a Java application programming interface for performing remote procedure calls for distributed computing. Because it is implemented in Java, it is platform independent. Similar to CORBA, it

generates the stub and skeleton classes for the client and server. Java interface is a natural construct for the interface definition.

There are two common implementations of the interface, the initial one to be implemented known as JRMP and a version compatible with CORBA. Usage of the term RMI may solely denote the programming interface or may signify both the API and JRMP, whereas the term RMI-IIOP, read RMI over IIOP, denotes the RMI interface delegating the most of the functionality to the supporting CORBA implementation. The original RMI API was generalised somewhat to support different implementations [106]. Additionally, work was done to CORBA, adding a pass by value capability, to support the RMI interface. Still, the RMI-IIOP and JRMP implementations are not fully identical in their interfaces.

2.5.5. DCOM

Distributed Component Object Model (DCOM) [125] is a Microsoft proprietary technology for software components distributed across several networked computers to communicate with each other. It extends Microsoft's COM, and provides the communication substrate under Microsoft's COM+ application server infrastructure. It has been deprecated in favour of Microsoft .NET.

In terms of the extensions it added to COM, DCOM had to solve the problems of marshalling (serialising and desterilising the arguments and return values of method calls "over the wire") and distributed garbage collection (ensuring that references held by clients of interfaces are released when, for example, the client process crashed, or the network connection was lost) [63]. DCOM was a major competitor to CORBA. Proponents of both of these technologies indicate that they will become the model for code and service-reuse over the Internet. However, the difficulties involved in getting either of these technologies to work over Internet firewalls, and on unknown and insecure

machines, meant that normal HTTP requests in combination with Web browsers won out over both of them. Microsoft, at one point, attempted and failed to head this off by adding an extra http transport to RPC [87] called "ncacn_http" (Network Computing Architecture, Connection-based, over HTTP).

In a world, DCOM is another object technology integration environment supported only on Microsoft Windows platforms. DCOM-based applications can take full advantage of the existing Microsoft services such as security and transactions. Some software vendors, migrate DCOM to other platforms such as UNIX so that, DCOM may eventually become a truly cross-platform technology.

2.5.6. Peer to Peer

A Peer-to-Peer (or P2P) [108] computer network is a network that relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. P2P networks are typically used for connecting nodes via largely ad hoc connections. Such networks are useful for many purposes. Sharing content files (see file sharing) containing audio, video, data or anything in digital format is very common, and real time data, such as telephony traffic, is also passed using P2P technology [100].

In P2P computing, machines share data and resources, such as spare computing cycles and storage capacity via the Internet or private networks. Machines can also communicate directly and manage computing tasks without using central servers. It is defined as a class of applications that takes advantage of resources storage, cycles, content, human presence available at the edges of the Internet [41].

A pure P2P network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both clients and servers to the other nodes on the network. This model of network arrangement differs from the client-server model where

communication is usually to and from a central server. A typical example for a non P2P file transfer is an FTP server where the client and server programs are quite distinct, and the clients initiate the download/uploads and the servers satisfy these requests.

2.5.7. Jini

Jini [4] is a network architecture for the construction of distributed systems where scale, rate of change and complexity of interactions within and between networks are extremely important and cannot be satisfactorily addressed by existing technologies [37]. It is a Java-based infrastructure developed by Sun Microsystems that can provide all the services necessary to support parallel and distributed applications [10]. Jini is primarily concerned with communications between devices. It is designed to provide a software infrastructure that can form a distributed computing environment, that offers network plug and play.

Jini technology provides a flexible infrastructure for delivering services in a network and creating spontaneous interactions between clients that use these services regardless of their hardware or software implementations.

Jini network technology is an open architecture that enables developers to create network-centric services (whether implemented in hardware or software) that are highly adaptive to change. Jini technology can be used to build adaptive networks that are scalable, evolvable and flexible as required in dynamic computing environments.

Integrate Grid technology with Jini technologies can make a great signification. Grid can help them become more powerful. The Grid is not only a computing paradigm for providing computational resources for grand challenge applications, but also an infrastructure that can bond and unify globally remote and diverse resources from disparate administrative domains. As such, the Grid can provide pervasive services to all users that need them.

2.5.8. Web Services

With the rapid global adoption of Business to Business (B2B) [123] and e-Commerce activities worldwide, the concept of the service aims at empowering the business process integration over the Web. To facilitate the use of the Web for business process integration and collaboration between trading partners, the service builds on the top of components referred to as Web services [21]. Such services offer specific business related functionality, reside in the application servers, can be programmatically integrated with other systems, and perform interactive tasks involving multiple steps on a user's behalf.

To allow for services to interoperate, a common industry standard, such as SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) and UDDI (Universal Description, Discovery, Integration) has been proposed. SOAP, which is encoded in XML and HTTP, is a message passing mechanism and serves as a uniform invocation mechanism between Web services. The WSDL describes the interface points of the Web services that are further indexed in searchable UDDI.

Chapter 3

Related Research

3.1. Software Evolution

3.1.1. Legacy System

First of all, the term software used in this thesis is to indicate source code, documentation and any related data. Specifically, the documentation part may include various specifications and designs of the existing system. The data part may include various related data such as input and output test data. The source code may include any programs written in one or more programming languages and integrated in a single system as well as any dependency files that are needed to compile and run the system.

A legacy system is an application program, which is currently a well-accepted and well-defined term within the software engineering community [151]. In 1995, Bennett defined legacy systems informally as, large software systems that we don't know how to cope with but that are vital to our organisation [13]. Legacy system is a computer system or application program which continues to be used because of the cost of replacing or redesigning it, despite its poor competitiveness and compatibility with modern equivalents. The implication is that the system is large, monolithic and difficult to modify.

Legacy systems are usually maintained by as many as hundreds of programmers for a number of years, and while many changes will be made to the software, the supporting documentation may not always be kept up to date. Legacy systems may have some of these characteristics:

- High maintenance costs.
- Complex software.
- Obsolete support software.
- Obsolete hardware.
- Lacking technical expertise.
- Business critical.
- Backlog of change requests.
- Poor documentation.
- Embedded business knowledge.
- Poorly understood by maintainers.

Despite these problems, organisations can have compelling reasons for keeping a legacy system, such as:

- The costs of redesigning the system are prohibitive because it is large, monolithic, and complex.
- The system requires close to 100% availability, so it can not be taken out of service, and the cost of designing a new system with a similar availability level is high.
- The way the system works is not well understood. Such a situation can occur when the designers of the system have left the organisation and the system has either not been fully documented or such documentation has been lost.
- The user expects that the system can be easily replaced when it becomes necessary.

- The system works satisfactorily, and the owner seems no reason for changing it – or in other words, re-learning a new system would have a prohibitive attendant cost in lost time and money.

3.1.2. Software Evolution and Reengineering

Software evolution is the process of adapting an existing software system to conform to an enhanced set of requirements. Software reengineering [150] is software evolution performed in a systematic way [124]. In particular, Chikofsky and Cross define reengineering to be "the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form". Altering existing systems comprises the majority of all software development time [7] and expense, and evolution comprises the majority of system alteration (maintenance) activities [155].

The major difference between initial development and evolution has to take into account the existing version of the system being evolved [121]. The important concerns including making sure that the new requirements are consistent with those of the existing version, trying to maintain control of the architecture of the system, understanding the code of the current version, and suggesting how the enhancement might be made while maintaining the conceptual integrity of the design. It will help the legacy system to migrate to a new language, platform, operating system, hardware; migrate to a new software development paradigm; increase maintainability and integrate with other systems [29].

Software evolution is the process of conducting continuous software reengineering. Reengineering implies a single change cycle, but evolution can go on forever. In other words, for a large extent, software evolution is repeated software reengineering.

Software reengineering technology is a practical solution for the problem of evolving existing computing systems. Due to the rapid development of computer hardware and software, the demands and costs of software changes are increasing continuously.

The phases of software reengineering include:

Reverse engineering is the process of analysing a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or higher level of abstraction [29].

The reverse engineering includes the following research areas:

Reengineering requirements involves the planning phase where the reengineering objectives are clearly identified within the context of the selected legacy system that needs be migrated [93].

Analysis involves a wide variety of source code analysis and abstraction models to assess the technical, functional and architectural aspects of the existing systems [115].

Positioning focuses on restructuring a system to enhance its qualities, or making the program more readable without changing its external behaviour.

Re-documentation is a form of restructuring a system with a semantically-equivalent representation in a different view in order to facilitate understanding.

Design recovery [31] aims to identify meaningful higher-level abstractions of a system and associates code segments with specific functionality.

Forward engineering is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system. It aims to improve a software system by considering new functional and non-functional requirements for the migrant system. Forward engineering follows a sequence of activities, including elicitation of new requirements, transformation of the system, and finally deployment of the system in its new environment.

3.1.3. Software Maintenance

Despite the large expenditure, little is known about the empirical nature of software maintenance, in terms of its effect on the artefact, on the process and on the software engineers and users. The first vista in the research landscape is therefore:

- To gain more empirical information about the nature of software maintenance, in terms of its effect on the software itself, on processes, and on organisations and people.
- To express such understanding in terms of predictive models which can be validated through experiment. The models may inform both the technical and business facets of maintenance.
- To explore and formalise the relationships between technology and business models.
- To understand how such models may be exploited in an industrial context.
- To establish accepted evaluation procedures for assessing new developments and processes, in terms of the implications for maintenance, especially in an industrial context on large scale applications.

Software maintenance is defined in IEEE Standard 1219 [78] as: the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

A similar definition is given by ISO/IEC [79], again stressing the post-delivery nature: The software product undergoes modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software product while preserving its integrity.

Software maintenance is often bug corrections or small functional enhancements and should never involve major structural changes [31]. Maintenance is required to support the evolution of any system, but it does have limitations:

- Legacy systems have been developed to be efficient rather than maintainable.
- Documentation is often inconsistent.
- Competitive advantage derived from adopting new technologies is seriously constrained. Net-centric computing or graphical user interfaces, for example, are not typically considered as a maintenance operation.
- Maintenance costs for legacy systems are increasing with time. Finding needed expertise in out-of-date technologies becomes increasingly difficult and expensive.
- Assessing the impact of changes is difficult. Often, the compound impact of many small changes is greater than the sum of the individual changes due to the erosion of the system's conceptual integrity. Information systems tend to expand with time as efforts to remove unused code are seldom funded. Modifying a legacy system to adapt it to new business needs becomes increasingly difficult.

3.1.4. Software Restructuring

Software restructuring is the transformation from one software representation form to another at the same relative abstraction level, while preserving the subject system's external behaviour (functionality and semantics) [29].

A number of restructuring techniques have been presented in [5]. These are organised in five levels: namely, source code, documentation, programming environment, coding standards, and management.

In the code level restructuring, some approaches modify the textual code to a standard style, for example, pretty printing and replacing poorly structured software with reusable components.

In the documentation level, some approaches take existing documentation, update it, and make comments more accurately, often without creating new forms of documentation. Other approaches, called system modularisation techniques, focus on the decomposition of a system into logically meaningful modules.

These techniques do not directly reconstruct the target software, but they aim to increase the ability of a programmer to deal with the software in the future maintenance tasks.

Software restructuring allows for source code transformations to be applied in order to facilitate program understanding and consequently alter the system being reengineered in order to increase its maintainability.

3.2. Software Reengineering

As an emerging trend in Net-Centric Computing technology [27], Grid focuses on large-scale resource sharing, innovative applications and high performance orientation. The sharing relationships may be static and long-lived or highly dynamic. With increasing adoption of distributed systems and Web based applications, more and more existing applications turn to legacy systems. Grid technology can be applied to achieve enterprise-wide applications integration. Adapting Grid to evolving legacy systems could provide a rich set of capabilities. It allows organisations to use numerous computers to solve problems by sharing computing resources [84].

Evolve legacy software systems to Grid platforms could receive significant attention in the future years. This approach aims to migrate selected parts of legacy systems to Grid platforms and designs. With the properties, such as information hiding, inheritance and

polymorphism inherent in Grid oriented designs, essential parts of such a reengineered system can be reused or integrated with other applications by using Grid computing technologies, or enterprise integration solutions.

In the related literature, approaches to migrate legacy systems to a new network-centric, distributed environment can be divided into three categories: re-development, restructuring and combination [153].

In re-development approach, the legacy system is redeveloped within the context of a client/server or multi-tier architecture. Re-development implies that all of the existing applications must be reprogrammed in a different programming language. Since the original system has been maintained for many years and encapsulates significant logic is not fully documented in most cases, re-development is the most risky of all approaches to apply.

The second approach focuses on restructuring a legacy system. The application programs must be modularised and restructured before they can be individually migrated. In addition, the data, accessed by the application programs, may be required to be migrated from its present form to distributed relational databases [142]. This is a more feasible solution, but requires that the whole control flow, input/output processing logic. The effort required and the risks involved are often high enough to discourage even the most eager client/server architecture proponent [127].

The third approaches is a combination of wrapping and entire system comprehension, which is called grey-box approach. Encapsulation provides the means by which an existing software component can be accessed in a transparent to the client way, via a separate wrapper object. This is a more promising alternative because it essentially provides an encapsulation layer to bridge the gap between existing legacy software application or components and new applications. It is also economic and practical to

extract some legacy functionality as components and gain supports from component based software engineering.

3.2.1. Component-Based Software Engineering

Component Based Software Engineering (CBSE) [16] is a process that aims to design and construct software systems using reusable software components [20]. The component paradigm starts with the assertion of an assembly-oriented view of software engineering [23], building software applications by wiring together the ports [66] and connectors of a set of pre-fabricated parts (components) within a component context [107]. This paradigm is an evolution of the notion of the object paradigm on the following aspects: an object having identity, state, and behaviour; a component exposing services, contracts, and manners and most importantly, it is configurable without requiring intrusive changes for using it.

In order to exploit the high-level information for scheduling in a Grid context [11], it is necessary to adopt the component based design pattern [133] to provide the portability, mobility and high performance while retaining the facility to maintain and export the required component meta-data at run-time.

Software components must conform to a component model, which is used to achieve component composition and assemble application from components [72]. For example, the data exchange model allows users and applications to interact and transfer data. Another example is the underlying object model, which allows the interoperation of components developed in different programming languages that reside on different platforms. The industrial standard component models of achieving this are CORBA, EJB and DCOM.

3.2.2. Software Clustering

Clustering [117] is a common technique for statistical data analysis, which is used in many fields, including machine learning, data mining, pattern recognition, image analysis and bioinformatics. Clustering is the classification of similar objects into different groups [34], or more precisely, the partitioning of a data set into subsets (clusters) [69], so that the data in each subset share some common trait (often proximity according to some defined distance measure) [152].

Clustering algorithms can be hierarchical or partitional. Hierarchical algorithms [94] find successive clusters using previously established clusters, whereas partitional algorithms determine all clusters at once [116]. Hierarchical algorithms can be agglomerative (bottom-up) or divisive (top-down) [146]. Agglomerative algorithms begin with each element as a separate cluster and merge them in successively larger clusters. Divisive algorithms begin with the whole set and proceed to divide it into successively smaller clusters.

Clustering analysis is to group large mounts of entities in a dataset into clusters according to their relationship and similarity [35]. Software clustering technique is applied to capture reusable legacy code segments [86], which is independent and loose coupling [149]. It is also applied in program understanding area, such as re-modularisation [122] and component recovery [143]. It is very useful to renovate legacy systems. Clustering methods can be used to identify objects in procedure based legacy systems [70].

3.2.3. Program Slicing

Mostly, legacy systems are huge and complex. Slicing a system into many small parts may receive many benefits, including: more flexibility and more reliability, as well as more reusability. Program slicing is a software maintenance technique used to identify all

program code that can in any way affect the value of a given variable. The following paragraph informally describes this computation. It is an established technique for reverse engineering and other analyses like testing or debugging. It is available in research prototypes and even in commercial products [6].

Program slicing [139] was defined by Weiser in 1979 as an presented an approach to compute slice by computing consecutive sets of indirectly relevant statement according to data flow and control flow dependences [17]. Only statically available information is used for computing slices, this type of slice is referred to as static slice. Dynamic program slicing [40] can be produced for a given input by maintaining a runtime representation of the syntax tree and marking nodes as their corresponding constructs are executed [101].

There are two main approaches of slicing: The original slicing technique from Weiser [58] is based on traditional data flow analysis [148]; the other approach is based on program dependence graphs (PDG) [39]. Extensive evaluations of different slicing algorithms have not really been done yet for control flow graph based algorithms that some data reported by Atkinson and Griswold can be found in [25]. The only evaluation of program dependence based algorithms that the author is aware of has been conducted by Agrawal and Guo, who just compare two algorithms [68]. Slicing identifies statements in a program which may influence a given statement (the slicing criterion) [88], but it cannot answer the question why a specific statement is part of a slice [14].

3.2.4. Wrapping Technique

Wrapping is a practice that transforms a component's software interface from one form to another. These techniques aim on enhancing interoperability and system flexibility through metadata [104].

The process of wrapping involves different techniques depending on the accessible elements of a legacy system. In the best case scenario, accessing to legacy system source

code is available. In this case, it is possible to integrate the legacy system directly to the object wrapper code.

Wrapping technology is used to remove mismatches between the interface exported by a software artefact and the interfaces required by current integration practices. Wrapping technology can modernise legacy system at user interface, data, or the functional (logic) level [137]. The user interface (UI) is the most visible part of a system. UI wrapping improves usability and is greatly appreciated by final users. Data wrapping enables accessing legacy data using a different interface or protocol than those for which the data was designed initially. Data wrapping improves connectivity and allows the integration of legacy data into modern infrastructures. In contrast with data wrapping, functional wrapping not only encapsulates the legacy data, but also the business logic embedded in the legacy system.

UI wrapping consists of wrapping old, text based interfaces with new graphical interfaces. This technique can be extended easily, enabling one new UI to wrap a number of legacy systems. Screen scraping is effective for stable systems where the principle objective is to improve usability. However, the new system is inflexible and difficult to maintain as the legacy system.

Alternative, a XML server has been designed to bridge legacy system to XML in the XML-based B2B architecture. The XML server acts as the contact point between the corporate infrastructure and the rest of the world. The XML server communicates by various means with the internal infrastructures including ERP systems, databases, EDIs, etc. On the other hand, the server interoperates with external organisation by exchanging XML messages.

Data Wrapping: Database Gateway is a specific type of software gateway that translates between two or more data access protocols. It normally translates a vendor-specific access protocol into one of the standard protocols, such as Open Database Connectivity

(ODBC) and Java Database Connectivity (JDBC). Using a database gateway to access legacy data improves connectivity, enables remote access, and supports the integration of legacy data with modern systems. Integration is another data wrapping technology. It uses XML server [31] to communicate with the internal infrastructures and external organisation by exchanging XML messages. It acts as the contact point between the corporate infrastructure and the rest of the world. In addition, most of the commercial XML servers support a wealth of communication protocols and this enables cost-effective integration with the most usual legacy applications. Database replication is also used for data wrapping. It is the process of copying and maintaining database objects in multiple databases that make up a distributed database system. Database replication is often used to enable decentralised access to legacy data stored in mainframes. New applications using the data receive the benefits of local access to a modern database instead of the problems of remote access to an obsolete data repository.

CGI Integration: The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP (HyperText Transfer Protocol) or Web servers. Legacy integration using the CGI is often used to provide fast Web access to existing assets including mainframes and transaction monitors. CGI integration not only wraps the old user interface, but also communicates with the core business logic or data of the legacy system. It is more flexible than screen scraping because the new interface does not need to match the old user interface. However, it still does not fully address maintenance issues.

Object-Oriented Wrapping: The conceptual model of object-oriented wrapping is deceptively simple: individual applications are represented as objects; common services are represented as objects; and business data is represented as objects. In reality, object-oriented wrapping is far from simple and involves several tasks including code analysis, decomposition, and abstraction of the Object-Oriented (OO) model. Among the multiple technical difficulties involved in wrapping a legacy system, two have special

relevance: the definition of appropriate object-level interfaces and the need for integrated infrastructure services.

Component Wrapping: Component wrapping is very similar to OO wrapping, but components, in contrast with objects, must conform to a component model. This constraint enables the component framework to provide the component with quality services [28].

In [127], wrapping is classified into five different levels, namely, job level, transaction level, program level, module level and procedure level. At process level and transaction level, wrappers encapsulate a batch of legacy executive processes. Legacy applications are invoked through wrappers by creating the requested new process and directly deploying the corresponding service without prior knowledge of the corresponding legacy code. In the application level, the wrapper encapsulates only one process. By contrast, at the module level and procedure level encapsulation focuses on clear interfaces, restructured and re-modularised legacy code.

3.3. Software Reuse for Grid

3.3.1. Grid Middleware Oriented Migration

Middleware is reusable software that resides between the applications and the underlying operating systems, network protocol stacks, and hardware. Appling middleware technology in Grid network is an attractive way for re-scheduling tasks in distributed environment.

Grid middleware should enable new capabilities to be constructed dynamically and transparently from distributed services. In order to engineer new Grid applications it is desirable to be able to reuse existing software components and information resources and assemble and coordinate these components in a flexible manner. Partly because this

reason the Grid middleware simplify Grid programming by raising the level of abstraction and reducing the accidental complexities. With the support of process coordination [8] and process migration [99], various runtime load balancing schemes can be employed for improving the execution efficiency of Grid applications. It helps those long-running applications by relocating them at suitable times to prevent interruption due to system activities or the execution of other applications [27]. It also can help to relocate processes closer to the Grid point with data that they need to access.

Legion [145] was the first integrated Grid middleware architected from first principles to address the complexity of Grid environments. Just as a traditional operating system provides an abstract interface to the underlying physical resources of a machine, Legion was designed to provide a powerful virtual machine interface layered over the distributed, heterogeneous, autonomous, and fault-prone physical and logical resources that constitute a Grid.

[59] proposed an ecological network-based Grid middleware (ENGM), which is based on ecological network computing environment (ENCE). It provides a new computing and problem-solving paradigm by combining natural ecosystem mechanisms with agent technologies and supports desirable requirements of new Grid systems, namely scalability, adaptability, self-organisation, simplicity, and survivability.

Scalable Inter-Grid Network Adaptation Layers (Signal) [145] is a middleware architecture which integrates mobile devices with existing Grid platforms to conduct peer-to-peer operations through proxy-based systems.

The Grid Application Toolkit (GAT) [1] provides a unified simple programming interface to the Grid infrastructure, tailored to the needs of Grid application programmers and users. Its implementation handles both the complexity and the variety of existing Grid middleware services via so-called adaptors.

To facilitate transparent use of the high-performance Across Trophic-Level System Simulation (ATLSS) ecosystem-modelling package for natural-resource management, [145] developed a Grid service module. The module exploits Grid middleware functionality to process complex computation without requiring users to handle underlying issues.

NAREGI [140] aims to research and develop high-performance, scalable Grid middleware for the national scientific computational infrastructure. Such middleware will help facilitate computing centres within Japan as well as worldwide in constructing a large-scale scientific "Research Grid" for all areas of science and engineering, to construct a "National Research Grid".

3.3.2. Web Based System and Service Oriented System

Software evolution is an effective way to improve the performance of Web based systems. A few approaches have been proposed on extracting information from Web based system and reengineering them into service oriented systems.

[132] presents various issues and challenges in Web based systems development and maintenance, especially in the public domain. An evolutionary Web-based system architecture is presented for management systems, and some design and implementation tradeoffs are discussed as well.

[19] suggests applying conventional reverse engineering techniques such as code analysis and clone detection in order to reduce duplicated content and achieve maintainable Web based systems. The developed system includes an HTML parser and analysers that separate content from layout by integrating in HTML pages scripts for retrieving the dynamic data from a database.

Clustering can be used to produce cohesive groups of pages that are displayed as a single node in reverse engineered diagrams. In [140], Tonella proposed a cluster method which

is based on the automatic extraction of the keywords of Web pages. The presence of common keywords is exploited to decide when it is appropriate to group pages together. A second usage of the keywords is in the automatic labelling of the recovered clusters of pages.

Lixto [131] is a wrapper generation tool which is well suitable for building HTML/XML wrappers. Otherwise, [38] presents a tool-supported method to reengineer Websites, that is, to extract the page contents as XML documents structured by expressive DTDs or XML Schemas.

[131] and [103] presented an approach to WWW information integration, based on the development of a canonical domain model in XML and the wrapping of existing WWW applications with wrappers capable of communicating about entities in this common model with the applications and with an intermediary mediator.

[82] and [128] aims at developing a technique for wrapping existing Web applications with WSDL descriptions so that organisations with a presence on the "browser-accessible Web" can easily enable programmatic access their functionalities to other applications.

3.3.3. Toward Grid Services

Many approaches to the reuse and integration of legacy systems in a Grid environment have been proposed in previous researches. Most of previous studies are focused on the analysis or the maintenance of legacy applications, as well as a few studies tried to wrap the existing applications followed by the Grid service standards.

The Numerically Intensive Java project (NINJA) [103] shows there are no serious technical impediments to the adoption of Java as a major language for numerically intensive computing. In NINJA, language and compiler techniques are used to address Java performance problems. This type of approach is essential, if Java is to be adopted throughout the high performance computing community. Huang describes a

semi-automatic conversion from legacy C code into Java code in [71]. After wrapping the native C application with the JACAW (Java-C Automatic Wrapper) tool, MEDLI (MEdiation of Data and Legacy Code Interface) is used for data mapping in order to make the code available as parts of a Grid workflow.

Based on different principles, [85] describes the deployment of legacy code as Grid services in GEMLCA (Grid Execution Management for Legacy Code Architecture) without modifying the original legacy code. This approach only supports GT-3 services, which is based on Open Grid Services Architecture (OGSA).

Most previous researches are largely concentrated on the study of re engineering scientific applications using the Open Grid Services Architecture [85]. Furthermore, the solutions apply only to one specific programming language, such as FORTRAN in [24] and COBOL in [18].

With the development of Grid, WSRF, announced in 2004 is an improvement of OGSA, and it is a new way for manipulating "stateful resources" to perform Grid services. [48] compared OGSI to WSRF, and described the relationship between the concepts, mechanisms, and syntax defined by the OGSI and the proposed WSRF as well as the related WS-notification family of specifications.

[147] describes a design to achieve compliance with the WSRF specifications using Microsoft .NET technologies. WSRF.NET is the toolkit for implementing WSRF-compliant Web services that is built on top of the .NET platform and [147] describes an application of exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments.

[74] presented a detailed analysis of five different implementations of WSRF, including GT4-JAVA (the Java Web Services Core of the Globus Toolkit v4), pyGRIDWare (a Python WSRF implementation, which is also distributed with Globus Toolkit v4 as its Python Web services Core), GT4-C (the C Web Services Core of the Globus Toolkit v4),

WSRF::Lite (Perl-based WSRF implementation) and WSRF.NET (an implementation of WSFW and WS-Notification on the .NET Framework).

3.4. Summary

Grid has become a new research interest as it performs a seamless information processing system across distributed, heterogeneous, dynamic virtual organisations that the user can access from any location. Together with software evolution techniques, legacy software systems can be inherited, and the Grid application development can be much more effective as well.

As Grid is an emerging technique, there is not much work for integrating Grid technique with software evolution. Although current researches on software evolution have covered a number of areas and made significant advances, as well as the Grid application developing. From the legacy systems evolution perspective, few of them bridge software evolution and Grid development together. Most of these researches focus on the reusable resources identification from legacy systems, but they can not be integrated in the Grid system directly.

Chapter 4

Proposed Framework

Legacy software systems can not be simply discarded as they are critical to business they support and they encapsulate a great deal of knowledge and expertise about the application domain [13]. Affected by the Grid orientation trend, many existing software systems will turn into legacy systems. These legacy systems require Grid oriented reengineering, which can facilitate legacy system evolution in Grid service orientated computing environments. From economic aspects, a business is constantly reorganising, changing its boundaries and reconfiguring its activities. Grid oriented reengineering enables legacy systems to adapt to continuous changes in business logic and market requirements. From technical aspects, application integration towards Grid and service choreography will become common in Grid service oriented environment.

However, Grid is quite a new technology and so there is currently few active researches related to reengineering legacy system within Grid environment. Much more researches are necessary and significant in this area to leverage and extend legacy systems in Grid environment. As a result, there is increasing interests in migrating and reengineering legacy systems into the Grid environment.

4.1. Grid Oriented Evolution

The Grid can be identified as three generations: First generation systems involved proprietary solutions for sharing high performance computing resources. Second generation systems introduced middleware to cope with scale and heterogeneity, with a focus on large-scale computational power and large volumes of data. Third generation

systems are adopting a service-oriented approach, are metadata-enabled and may exhibit autonomic features [120]. This research aims to provide a solution to evolving legacy systems into Grid environment especially the Grid services environment. Two major tasks are implied in this statement: (1) understand and decompose legacy systems and (2) build Grid systems with the reusable legacy system components.

As an innovative technology, there are many research fields within Grid application area, such as: Grid computing, Grid services and semantic Grid framework.

Grid computing is the initial concept of Grid technology, which is emerged as a wide area distributed computing technique to solve the arduous and time-consuming scientific computing. Nowadays, it has been extended as a Net Centric Computing which enables users to collaborate securely by sharing processing, applications, and data across systems to facilitate collaboration, faster application execution, and easier data accessing.

With the development of Service Oriented Architecture (SOA) and Web services, Grid services technology has attracted more and more significance than ever. Grid services are basically Web services with improved characteristics and services. The most important improvement is that Grid services are stateful and transient services. It can remember what have been done from one invocation to another with the support of stateful resources. Grid services are good complementarities for Web services, and they can be performed as independence services environment as well.

The semantic Grid is an extension of the current Grid in which information and services are given well-defined meanings, better enabling computers and people to work in cooperation. It arises with the parallel development of Web services, semantic Web and Grid computing.

Due to the various performance of Grid technology, different solution should be developed to achieve the Grid oriented legacy systems evolution.

4.2. Evolution Process

In this research, an approach for evolving legacy systems into Grid services environment has been described, and it is extended for semantic Grid environment as well. Based on some reverse engineering technologies, the proposed approach allows legacy system to be reused as Grid components. These Grid components are dynamic which could offer more flexibility and reusability for various requirements from users and applications. The framework of the Grid service oriented legacy software systems evolution approach is shown as Figure 4.1. It may consist of multiple phases as follows:

- Reverse engineering techniques are used for program comprehension and design recovery. The legacy software systems are decomposed into a hierarchy of subsystems by defining relationships between the entities of the underlying paradigm of the legacy system. The legacy system decomposition is achieved by program slicing and software clustering techniques.
- Grid components are created by XML transformation and representation.
- Grid components are defined as stateful resources and deployed into Grid services environment.

The aim of this approach is to use legacy systems into Grid environment which enables the integration of legacy resources with Grid across distributed, dynamic environment and communities.

This research proposes a solution for migrating legacy systems using a Grid user interface to enable the dynamic activation and Grid resources discovery. This ability of the legacy system using Grid services will allow the enterprise to take advantage of the broad commercial support provided by Grid technology. The proposed approach makes legacy systems run in Grid environment as Grid applications and it also enable legacy systems to be used as distributed system for supplying more working ability.

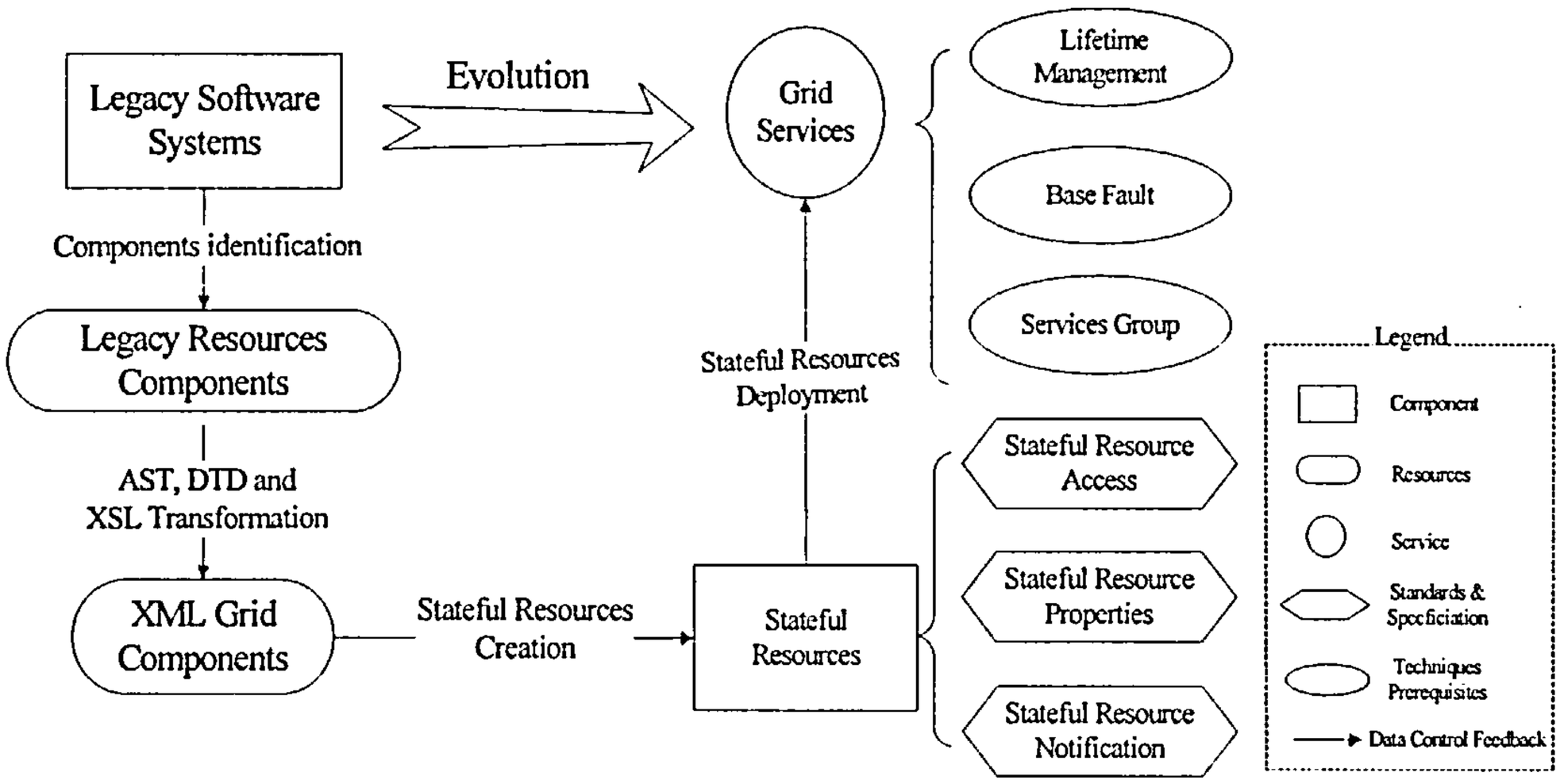

Figure 4.1. Framework of Grid Services Oriented Legacy Software Systems Evolution.

4.2.1. Component Identification

Component Based software engineering is a process that aims to design and construct software systems using reusable software components. The component paradigm starts with the assertion of an assembly-oriented view of software engineering, building software applications by wiring together the ports and connectors of a set of pre-fabricated parts (components) within a component context.

Component based development is different from previous approaches in its separation of component specification from implementation, and the division of component specifications into interfaces. The separation of component specification from implementation means that software components can be implemented in any programming language using any data storage mechanism. In particular, this means that existing software, which may not initially meet all the requirements of components, can be updated, so that it conforms to the component standard and becomes a valid component.

If a legacy system is selected as the application of component based reengineering approach, a component mining process will be performed. Reverse engineering techniques such as static program slicing and hierarchical agglomerative clustering are applied in this component mining process.

The component based Grid oriented approach is especially applicable to reengineering tasks with the following characteristics:

- Legacy systems have reusable and reliable functions embedded with valuable business logic.
- Reusable components extracted from a legacy system are fairly maintainable compared to maintain the whole legacy system.
- Some components of the target system run on different platforms or vendor products.

In this part, the program slicing technique is used to decompose system, understand program, eliminate dead code and make selected code segments function independently by component interface parameters determination and deep source code comprehension and analysis.

The software clustering technique is used to group large mounts of entities in a dataset and capture reusable legacy code segments into clusters according to their relationship and similarity from legacy systems, and create a hierarchical structure of these reusable legacy code segments.

Mostly, legacy systems are huge and complex. Using components based development approach to evolving legacy systems is less risky and highly transport and it will also bring more flexibility, expansibility, reusability and reliability. Also, this approach is low cost and easy to implement.

4.2.2. Grid Component Migration and Packing

After the legacy system components identification, these components have to be migrated for deploying in the Grid environment. This part presents the approach for migrating and packing the extracted legacy assets as Grid components.

For Grid components migration, the concepts of Grid middleware and process transfer have been used. While the target Grid application varies, the migration approaches are different with each other. The XML plays a great important role in all three kinds of evolution.

XML is used to describe the data structure and exchange information. Extensible Stylesheet Language (XSL) allows to build legacy codes which are insulated from changing the external format of both the legacy system and the calling application. And XSL is also used to manipulate XML from one structure into another. At last, XML schemas are employed to encapsulate and integrate legacy components into the Grid framework.

Both bottom up and top down approach can help to extract and represent legacy assets into XML format. The bottom-up approach utilises the concept of Grid component definition that denotes the syntactic structures of a programming language. The top-down approach examines the grammar of the specific programming language, and defines a standard logical structure.

In the proposed approach, a structure of the Abstract Syntax Tree (AST) has been defined to develop Grid components. By recursively traversing the hierarchy of the component entities, it is able to map the Grid component to a Document Type Definitions (DTD). Specifically, the tree hierarchical structures of the component are mapped to XML elements and attributes. Each node and edge in the AST is mapped to an XML element

tag. The attribute values of an AST node are mapped to the corresponding attribute values of the XML elements.

The Grid component and its corresponding DTD can be enhanced with information. Similarly, Grid component generalisations include the introduction of elements which relate to system constructs. In this context, the grammar of the programming language has been modelled to define the DTD and the organisation of the XML document that models the AST of a given source code fragment.

XML is increasingly finding acceptance as a standard not only for describing documents, but also as a data description language for all types of information. DTD are generated describing the characteristics of the data making the documents self contained and reusable as a data exchange format. Combined with client-side interpreted XSLT stylesheets, it can provide an approach to publishing data from relational databases on the Web.

This Grid components packing approach is based on XML representation and transformation. Once a software component has been extracted from a legacy system, or has been built as a new component, its interface can be extracted and represented in XML. In this thesis, the XML representation is not only finished by the component wrap, but also with the sources code analysis included, such as the using of AST, DTD and XSLT. At last, the legacy components are wrapped as XML components which could be further used in the Grid services environment.

4.2.3. Grid Service Integration

This section presents a framework that integrates the XML packaged legacy system components into the Grid service environment. For Grid service framework, stateful and service are eminent features. Based on the Web Services Resource Framework (WSRF),

the legacy resources are deployed in the Grid environment as stateful resources. These stateful resources can be isolated and integrated into components which can be integrated into Grid service oriented architectures. At last, they could be integrated into Grid services environment by exposing service oriented interface.

Deploying a Grid service can be divided as follow four steps.

1. Define the service's interface. This is done with WSDL.

2. Implement the service. This is done with Java.

3. Define the deployment parameters. This is done with WSDD and JNDI.

4. Deploy the services.

In the first step, the service properties and its interface are defined by WSDL. Then the implementation of the service is carried out by Java. In the third step, the WSDD and JNDI file define the deployment parameters including services registration and resources localisation. The last step is to deploy the Grid service. Different from the related studies, the proposed approach describes a method for defining the legacy resources as stateful resources, and builds the Grid services based on these reusable resources.

This kind of services can be used as ways of solving compute intensive problems. It can handle the computing which is much different from other types of computing in sharing managed resources among institutions. Also, it could supply extra working ability when it is necessary to be build a long term investment occurs.

In Grid services environment, service oriented architecture provides an interface for individual services to bind up as a new service, and break down integrated services dynamically. As legacy systems are restructured as Grid service components, dynamic services could be created in Grid environment.

Based on monitoring and discovery service which provides information about the available resources within Grid and their status, users submit new requirements by specified form to the Grid system through user interface. Resources management picks necessary services and creates a new Grid service for each special requirement. After finishing their particular operation, the services will be broken down again into original Grid service components. Then, these Grid service components are ready to use by other services. The integration of recovered legacy components in Grid services can be shown as Figure 4.2.

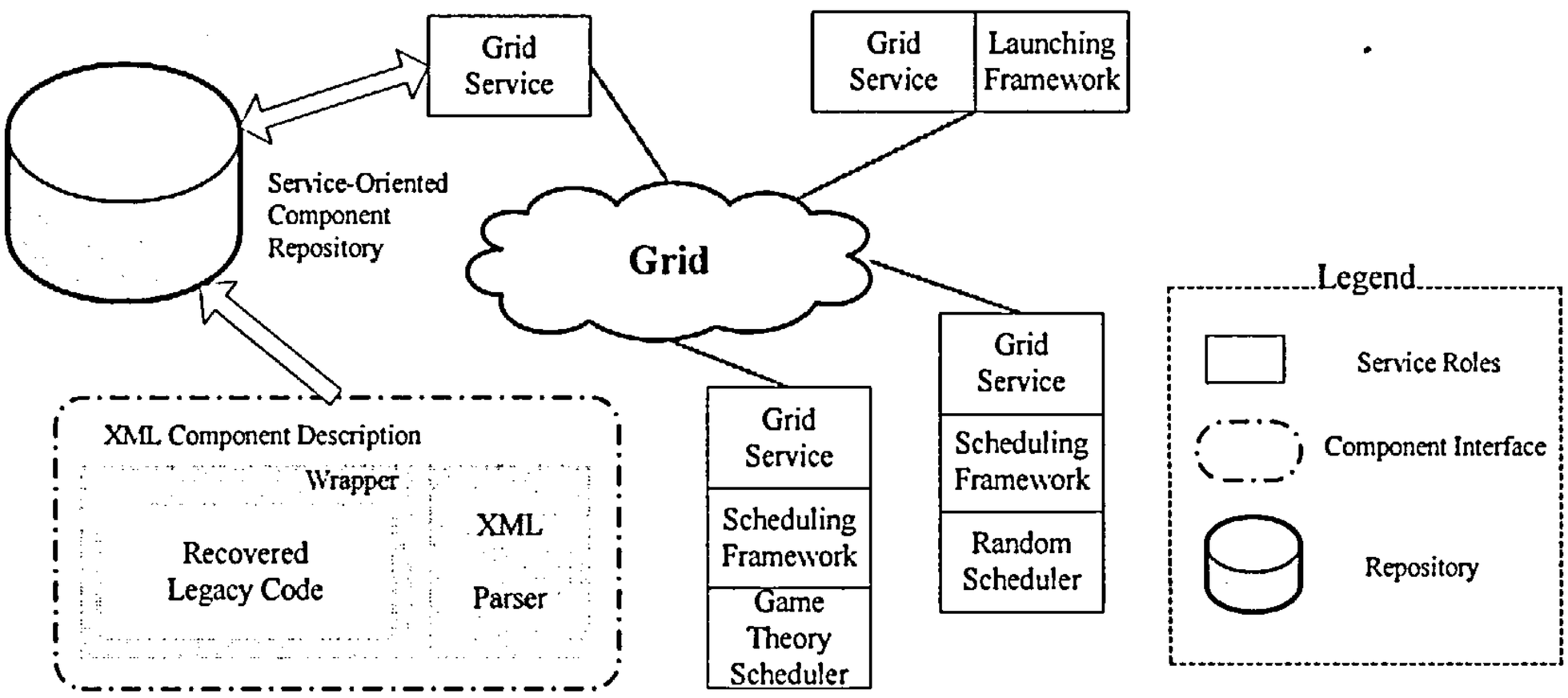

Figure 4.2. Legacy Resources and Grid Services Integration.

Various services will be created dynamically at different time for different user's needs. These services are transparent to users. Users just use services they want regardless where these services come from and how many individual services they integrated. Grid technology can connect services in a convenient way. A service can be exported to other communities, thus providing interaction between two or more isolated communities.

The idea of Grid service oriented legacy software system evolution also can be extended into semantic Grid structure. Although semantic Grid technology is not mature, and there is no common approach for building semantic Grid up to now, as it is an evolution from Grid computing and the semantic Web, many researches can be commenced based on these two techniques.

This thesis proposed an approach to reusing the legacy system assets in the semantic Grid framework. This approach is based on source code translation and reconstruction, component reusing and the semantic Grid framework retargeting. As the recovered legacy resources are strictly formatted and defined follow the semantic Grid criterion, with little revision or not, they could be easily applied into future standard semantic Grid systems.

When extending Grid service to semantic Grid environment, XML are employed to provide a simple and elegant frame for describing the properties of the reusable legacy system resources as Grid RDF data models which are based on a specific XML mark language RDF/XML. The RDF/XML representation and RDF schema description are key techniques in reusing recovered legacy components in semantic Grid framework.

4.3. Summary

This chapter presents the framework on Grid oriented evolution approach and evolution process of the framework. It aims to identify and extract components from legacy systems, and generates suitable Grid component for different Grid systems and integrates the evolved components in Grid environment.

Such a reengineering framework is composed of legacy system decomposition, component identification, Grid components composition and Grid environment

integration that are based on current feasible Grid technology. The following chapters will discuss the evolution process and the Grid platforms integration in detail.

Chapter 5

Component Identification for Use in Grid Environment

The popularity of the Internet as well as the availability of powerful computers and high speed network technologies as low cost commodity components is changing the way people use computers today (such as COBRA, J2EE, etc.). These technology opportunities have led to the possibility of using distributed computers as a single, unified computing resource, leading to what is popularly known as Grid computing.

Component based development (CBD) is the industrialisation of the software development process based on the assembly of prefabricated software components. Two basic ideas underlie CBD. Firstly, application development can be significantly improved if applications can be quickly assembled from prefabricated software components. Secondly, an increasingly large collection of interoperable software components will be made available to developers in both general and specialist catalogs.

A software component model is a system for assembling applications from smaller units called components. The system defines a set of rules that specify the precise execution environment provided to each component and the rules of behaviour and special design features that components must have in order to be considered true components.

Encapsulation is an important characteristic of distributed systems. This chapter defines a reengineering and migration method that identifies components from legacy system for use in Grid environment.

5.1. Component in Grid Oriented Evolution

The concept of Grid computing first started as a project to link geographically dispersed supercomputers, but now it has grown far beyond its original intent. In fact, many applications can benefit from the Grid infrastructure, including collaborative engineering, data exploration, high throughput computing and distributed supercomputing. For Grid applications development, there is indeed a need to smoothly, seamlessly and dynamically integrate and deploy autonomous software, and to provide glue in the form of a software bus for it.

A component is defined as a unit of software application composition with contractually specified interfaces and explicit context dependencies that can be developed, acquired, added to the system and composed of other independent components. Component interfaces determine the operations that a component implements, and the operations it uses from other components during its execution. A distributed component-oriented model is a framework for defining components and their interactions.

When reusing legacy assets in Grid platform, legacy system components should be retargeted as Grid components which make legacy assets to be interoperated and create secure, seamless and reliable Grid application access to vast IT capabilities. The main achievement of this work is to design and implement a concept of Grid components. Grid components are formed from legacy components. They wrapped legacy code and provided a standard interface for themselves to be used in the Grid platform. They could be deployed, recognised and moved such as to tackle fault-tolerance, load-balancing, adaptability to changing environmental conditions.

The idea of using component frameworks to deal with the complexity of developing Grid computing applications is becoming increasingly popular. Such systems enable programmers to accelerate project development by introducing higher level abstractions and allowing code reusability. They also provide clearly defined component interfaces,

which facilitate the task of team interaction. Such a standard will promote interoperability between components developed by different teams across different institutions.

5.2. Definition of Legacy Grid Component

Components were defined as "bits of software that can be replicated and, often with modifications, assembled repeatedly to form any number of applications". More flexible and adaptable components will only require configuration (rather than modification) before they can be reused. Another view is that "a reusable software component is a logically cohesive, loosely coupled module that denotes a single abstraction". High cohesive and low coupling are the basic features of components because of the variation in levels of abstraction, but it is also important to mention the context in which a component can be used. A further view is that a software component is a static abstraction with plugs. Here, "static" means that a software component is a long lived entity that can be stored in a software base, independent of the applications in which it has been used. "Abstraction" means that a component puts a more or less opaque boundary around the software in encapsulates. And "with plugs" means that there are well defined ways to interact and communicate with the component, such as parameters, ports, and messages.

To retrieve reusable resources from legacy systems for migrating to Grid environment, a term Grid component is introduced. Grid component extends the concept of component, it brings the component based development approach to the Grid research area. The concept of Grid component gives a way for evolving legacy systems into Grid environment.

Grid components build with legacy system components to support distributed computing application in Grid to couple resources that can not be replicated at a single site, or may be globally located for other practical reasons. These are some of the driving forces

behind the foundation of global Grid. In this light, the Grid allows users to solve bigger or new problems by pooling together Grid components resources that could not be coupled easily before. Grid components are formed of distributed subcomponents which are wrapped with legacy code. They can be deployed but further reconfigured and moved to tackle fault-tolerance, load-balancing, adaptability to changing environmental conditions.

Grid components extend the principles of components by strengthening the role of the XML interface and by adding the separate notion of component specification for Grid integration. A clear Grid component specification describes what it does. A major part of a Grid component specification is the definition of Grid component interfaces. It is a definition of a set of behaviours that can be offered by a component object.

A Grid component is an independent, encapsulated and physical part of a Grid system. It may be developed, tested and deployed in complete isolation to other Grid components within the Grid system. It looks like a coherent and configurable software package, independent of the Grid applications in which it has been used, with well defined interfaces in different contexts to interact and communicate with other Grid components, in order to compose a Grid system.

This chapter concentrated on identify component from legacy system for using in Grid environment. The Grid component migration and packing will be presented in the following chapter. The Grid oriented component based software reengineering process should contain steps of identification, classification, storing, retrieval, adaptation, and composition, such as the following:

- Mine components from the legacy systems.
- Wrap up components with well defined interfaces.
- Store the components in a component library.

- Build new reusable components if needed.
- Develop new systems by integrating components.

5.3. Design

Software systems evolve over time, as a result of requirement changes. The resulting systems tend to have a rich and complex structure, which is highly coupled. Due to long term maintenance and evolution, the documentations may not be able to reflect the actual structure of legacy systems. It is obvious that one can not maintain, reuse or redevelop a piece of software unless one is absolutely sure about the functionality of that software. Whether for forward engineering, or for reverse engineering, the ability to reuse, or recycle existing software components should reduce the complexity and cost of the task in hand. In many cases, effective partitioning or re-partitioning is needed to decompose legacy systems with a high cohesion and low coupling principle.

Component based development is different from previous approaches in its separation of component specification from implementation, and in the division of component specifications into interfaces. The separation of component specification from implementation means that software components can be implemented in any programming language using any data storage mechanism. In particular, this means that existing software, which may not initially meet all the requirements of components, can be updated, sometimes very simply, so that it conforms to the component standard and becomes a valid component.

A component is more packaged than an ordinary object. Components are larger than classes. They can be in any programming language, can include their own metadata, are assembled without programming, and need to specify what they require to run. Compared to objects, components are larger sized, physical entities, instead of conceptual entities, and supporting encapsulation with defined interfaces.

The assumption is that it will be used in many contexts that are unknown to its own designers. Components are identified by their interfaces. An interface should be defined in different context to interact and communicate with other components. The term black box conveys the idea of a component whose internal workings are hidden. The importance of component is the ways in which it interacts with other components over some well defined interface.

5.4. Domain Analysis

Analysing legacy systems is used to understand the current architecture and developing a strategy for mining and reusing existing assets. Mining involves rehabilitating parts of an old system for use in a new system. Analysis is crucial because many legacy systems do not have clear specifications, and the architecture of legacy system should be understudied as well as the approaches of building theses systems.

Legacy system evaluation reveals the current status of a legacy system and specifies in which phase of lifecycle it is. Domain analysis is a process in which information related to a system being developed is identified, captured and organised with the intention of making it reusable for new systems [113]. It is the process of analysing related software systems in a domain to find their common and variable parts. The goal of the domain analysis is to identify the document requirements on a set of systems in the same application domain.

The domain analysis process can be carried out in two steps, sub-domain identification and analysis of the selected sub-domain. The domain analysis process is targeting on the problem domain. The sub-domain identification determinates the boundaries of sub-domains and decomposes the whole problem domain.

The further sub-domain analysis focuses on modelling a particular part of the problem domain. The results of the whole domain analysis are summarised as a domain model.

Based on this domain model, some business functions are identified to be valuable and reusable and needed to be provided.

In this way, some logical services are summarised. These logical services are somewhat between the theoretical, ideal services that exactly support the sub-domain functionality and the set of existing physical software components.

5.5. Legacy System Decomposition

5.5.1. Decomposition Strategies

Decomposing a program entails the identification and the reorganisation of different program components. Basically, these components can be distinguished in interface components, application logic components, and database components. Interface components correspond to code fragments clustered around I/O statements on legacy assets. A database component might be any piece of code clustered around I/O statements. Application logic components correspond to code fragments implementing business rules.

Program slicing is a method for automatically decomposing a program by analysing its control and data flow. The proposed research focused on static slicing. Based on the original definition of Weiser, a static program slice S consists of all statements in program P that may affect the value of variable v at some point p.

The slice is defined for a slicing criterion C=(x,V), where x is a statement in program P and V is a subset of variables in P. A static slicing includes all the statements that affect variable v for a set of all possible inputs at the point of interest. Static slices are computed by finding consecutive sets of indirectly relevant statements according to data and control dependencies.

Program sling is an established technique for reverse engineering. With the help of slicing technology, legacy systems can be divided into some concerned program parts and discard the useless assets. Program sling is desirable to extract legacy system rules at high levels of abstraction of the program. Traditional, it is an established technique for reverse engineering. For the proposed approach of Grid oriented evolution, program slicing are used to decomposes legacy system, understand program, eliminate dead code and make selected code segments function independently.

This section deal with software systems composed of programs each of which may comprise all types of components and propose a technique to decompose them. The proposed approach exploits static analysis and program slicing techniques to identify the set of program statements that contribute to implement database and user interface components. More specifically, slicing is used to identify all the statements and predicates that implement operations, and these statements could be used to define the components to form new Grid system. They are reengineered as a Grid oriented style by extracting and encapsulating in separated subroutines code fragments implementing database components and application logic components.

5.5.2. Slicing Rules

There are two main approaches to slicing: The original slicing technique from Weiser is based on data flow and control flow analysis; the other approach is based on Program Dependence Graphs (PDG). The control flow analysis is adopted in the proposed approach for program representation and static analysis about programs at the intraprocedural and interprocedural levels.

A Control Flow Graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution. Directed edges are used to represent jumps in the control flow. In most presentations, there are two specially

designated blocks: the entry block, through which control enters into the flow graph, and the exit block, through which all control flow leaves.

A control flow graph for program *P* is a graph in which each node is associated with a statement from *P* and the edges represent the flow of control in *P*. Let *V* be the set of variables in *P*. With each node *n* (each statement in the program and node in the graph) associates two sets: *REF(n)*, the set of variables whose values are referenced at n, and *DEF(n)*, the set of variables whose values are defined at *n*.

Computing a slice from a control flow graph is a two step process: at the first step, requisite data flow information is computed and then this information is used to extract the slice. The data flow information is the set of relevant variables at each node n. For the slice with respect to <s, v>, the relevant set for each node contains the variables whose values affect the computation of v at s. The second step identifies the statements of the slice, which include all nodes (statements) n that assign to a variable relevant at n and the slice taken with respect to any predicate node that directly controls n's execution.

Preparing a program for Grid oriented migration requires the decomposition of the legacy systems that contains statements of both interface and database components. There are differences in the level of complexity of the decomposition techniques due to the target Grid application style that is chosen.

In the proposed approach, rules of slicing and chopping are based on dividing the system into independent, stable and high reusable parts. Each part can be evolved and served as Grid service resources. Meanwhile, these parts could be connected with each other flexibly and effectively. This requirement makes sure that all parts can be integrated as a whole service as the system without slicing.

5.5.3. Slicing Algorithms

The program slicing in this thesis refers to software systems written in procedural languages, such as C. Theses systems are typically composed of a set of programs that is related through external calls and can be represented by a graph whose nodes represent the programs and edges depict the call relation between programs. The call relation on the subroutines of a program can be represented by a graph whose nodes correspond to the program subroutines and edges depict the internal calls.

A control flow graph is a directed graph, where each node represents a predicate or a statement and each edge represents a transfer of the control between statements. Control dependencies can be represented by a control dependence graph, which is a directed graph containing the same set of nodes as the control flow graphs and whose edges depict the control dependence relation.

Control dependence is usually defined in terms of post-dominance. A node *i* in the CFG is post-dominated by a node *j* if all paths from *i* stop pass through *j*. Control dependences in programs could be determined by the control flow. For example, programming with structured control flow, control dependences can be determined in a simple syntax directed manner: the statements in the branches of an "if" or "while" are control dependent on the control predicate. Based on this approach, each program subroutine can be depicted by control flow graph and a control dependence graph.

For Slicing Flow Graphs of Straight Line Programs

Straight line code contains only assignment statements executed one after the other. For such code, the additional slices with respect to predicate nodes are not required. The slicing begins by assuming that expression evaluation does not alter the values of its operands.

The relevant sets may be viewed as a flow of sets of variables; the slice is the set of statements that disturb this flow. If no relevant variables are defined at a statement, then the relevant set flows through unperturbed. On the other hand, if a relevant variable is defined then the statement is added to the slice.

So the slicing algorithm of straight line programs can be presented as:

```
Assume expression evaluation;
    View relevant sets as flow of sets of variables;
            If (no relevant variables defined)
                {relevant set flows through unperturbed};
          else {the statement is added to the slice}
```

For Interprocedural Slicing

Slicing across procedures complicates the situation due to the necessity of translating and passing the criteria into and out of calling and called procedures. The proposed interprocedural slicing algorithm for identifying the set of statements contributing to implement components is based on the analysis of the control flow information of legacy system summarised in control flow and control dependence graphs.

Taking the control flow graph and control dependence graph of a legacy system as inputs, the sets of statements and predicate nodes in the Graph as well as the sets of program nodes in the Graph can be returned. In this way, the subroutines of all the programs in the legacy system would be analysed, and each program is analysed only after the programs with calls have been analysed. Similarly, the subroutines of a program are visited according to the partial order induced by the reverse internal call.

For each subroutine, the algorithm computes the initial sets of control flow graph that correspond to I/O statements. Then, the control dependencies are backward traversed, starting from the initial set of nodes; all nodes reached during this transitive closure are

included in the slice. If the initial set returned a subroutine is not empty, the subroutine, and the program that contain it, might cause the execution of I/O statements, and therefore they are respectively inserted in the sets of subroutine node and program node. The sets of statements and predicate nodes are also updated with the nodes in the current slice. The slicing algorithm of interprocedural slicing can be simply presented as:

```
  Define control flow graph and control dependence graph of a legacy system as inputs;
Analysis the subroutines of all the programs; {
        computes the initial sets of control flow graph;
        backward traverse the control dependencies;
        include all nodes reached in a slice;
        };
 Analysis each program in legacy system
```

For one variable, a decomposition slice has been built, which is the union of certain slices taken at certain line numbers on the given variable. Then the other component of the decomposition will also be obtained from the original program.

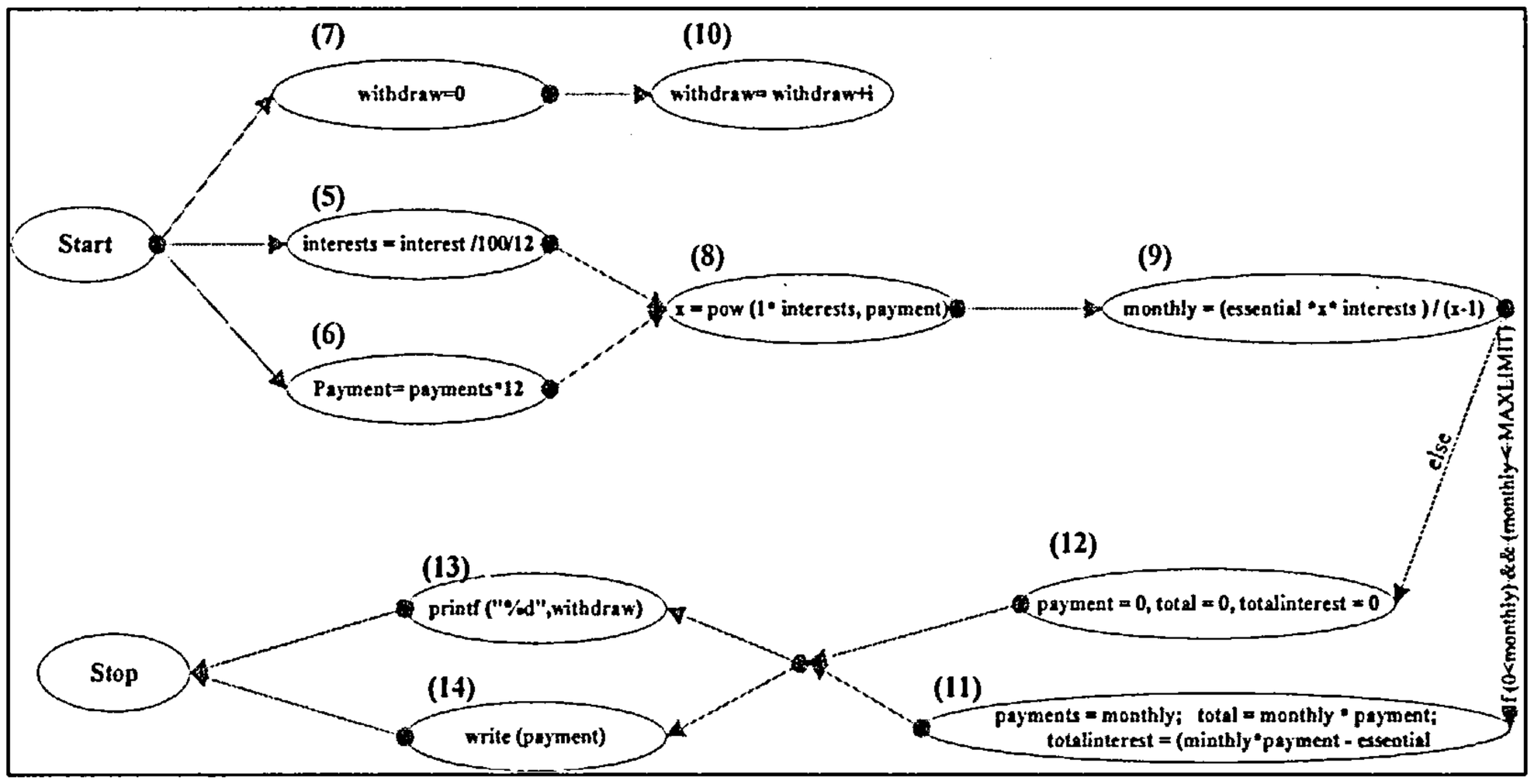

Figure 5.1. An Example of a Control Flow Graph.

These complements are constructed in such a way that when certain statements of the decomposition slice are removed from the original program, the program that remains is the slice that corresponds to the complement of the given criteria, with respect to the variables defined in the program. Thus the complement is also a program slice.

This section describes a solution to slice legacy problem using a control flow graph as an intermediate representation. It considers a progression of harder slicing problems beginning with slicing straight line programs, then considering procedures. Figure 5.1 shows an example of a control flow graph. The detail of its creation will be presented in the Chapter 9.

5.6. Using Hierarchical Cluster to Create Hierarchical Decomposition of Legacy Systems

Software systems evolve over time, as a result of requirement changes. The resulting systems tend to have a rich and complex structure, which is highly coupled. Due to long term maintenance and evolution, the documentations may not be able to reflect the actual structure of legacy systems. In many cases, effective partitioning or re-partitioning is needed to decompose legacy systems with a high cohesion and low coupling principle. Software clustering technique groups large mounts of entities in a dataset into clusters according to their relationship and similarity. It is applied to capture reusable legacy code segments, which are independent themselves.

Cluster analysis has been of long-standing interest in statistics, numerical analysis, machine learning and other fields. Current cluster analysis offers a wide range of techniques for identifying underlying structures in large sets of objects and revealing relationships between objects or the classes of objects.

A component is then nothing more than an object or collection of objects that obey the rules of the component architecture. A component framework is the software environment that provides the mechanisms to instantiate components, compose them, and use them to build applications. The execution environment of the component architecture which provides a component instance is often called component container.

From the point of view of Grid environment oriented programming, components are good candidates for modelling units of distribution because they encapsulate attributes and methods to act as independent entities communicating through passing messages.

The motivation of migrate legacy systems using Grid technologies is more than the obvious advantage of being able to use remote computational resources. For many legacy systems, reengineering of the legacy software to increase maintainability also increases the potential to use modern tools and techniques.

In the literature review, there exist several different clustering algorithms, with different properties. Hierarchical algorithms do not produce a single partition of the system. Their output is rather a tree, with the root consisting of one cluster enclosing all entities, and the leaves consisting of singleton clusters.

At each intermediate level, a partition of the system is available, with the number of clusters increasing while moving downward in the tree. Divisive algorithms start from the whole system at the tree root, and then the system is divided into smaller clusters, attached as tree children. Alternatively, agglomerative algorithms start from singleton clusters and join them together incrementally. Optimising algorithms produce a single partition of the system, obtained by some heuristics that aims at maximising a clustering quality measure.

Partitional algorithms determine all clusters at once, whereas hierarchical algorithms can be agglomerative (bottom-up) or divisive (top-down). The proposed approach in this thesis has preferred hierarchical algorithms for component identification in legacy

systems over optimising algorithms because they support a manual refinement of the clustering granularity and they are more suitable for further restructuring as Grid component by their hierarchical structure. After selecting a clustering tree level, the user can move upward or downward in the tree, if the clusters at the selected level are too specific (some codes to be migrated are missing) or too general (migration candidates do not share a recognisable template).

Most clustering techniques presented in the literature utilise certain criteria to decompose a system into a set of meaningful modular clusters. Such criteria attempt to achieve a cluster with low coupling, high cohesion, interface minimisation and sharing of neighbour resources. In the context of the Grid oriented migration, this thesis strives to produce clusters that assemble the maximal size of source code entities that are related to a component candidate.

5.6.1. Similarity Between Entities

When two clusters are joined or one cluster is split up in two smaller clusters, the similarities between the newly formed cluster(s) and the previously existing ones have to be calculated. When a similarity measure was used that can be applied to sets of entities these similarities can be computed from the original data. Otherwise the computation is done by means of an updating rule. An updating rule takes the similarities (and possibly some additional information like the number of entities in each cluster) from the previous step of the algorithm as input and computes a value which serves as the new similarity measure.

Before applying hierarchical agglomerative clustering analysis, the similarity between two entities and a clustering algorithm must be defined. Three common methods for defining similarity between clusters are the single-link method, the complete-link method, and ward's method.

The following discussion of updating rules assumes that two clusters, A and B, have been joined. The similarity between an already existing cluster C and this new cluster is computed.

An often used updating rule is the single linkage rule or nearest neighbour updating rule:

$$\text{Single link}(C, AUB) = \text{MAX}(\text{sim}(C, A), \text{sim}(C, B))$$

So cluster C is as similar to AUB as it is to the most similar of the old clusters.

The complete linkage rule or furthest neighbour rule takes the similarity with the least similar old cluster to be the new similarity:

$$\text{Compl link}(C, AUB) = \text{MIN}(\text{sim}(C, A), \text{sim}(C, B))$$

Unweighted pair-group method using arithmetic averages (UPGMA) is a simple bottom-up data clustering method used in bioinformatics for the creation of phylogenetic trees. It was initially designed for using in protein electrophoresis studies and is not a well-regarded method for inferring phylogenetic trees unless the constant-rate assumption (molecular clock hypothesis) has been tested and justified for the data set being used.

In the single link method, the similarity between two clusters C1; C2 is the maximum of the similarity between a pair d1; d2 where d1 2 C1 and d2 2 C2. Thus, two clusters are similar if some pair of members are similar. In the complete link method, the similarity between two clusters C1; C2 is the minimum of the similarity between a pair d1; d2 where d1 2 C1 and d2 2 C2. Thus, two clusters are similar if every pair of members is similar.

In ward's method, the pair of clusters that are considered to be closest together among all clusters (and hence are merged in a step of a hierarchical clustering algorithm) is the pair whose merger minimises a certain sum of squares error based on distances from centroids of the clusters. The complete link method is usually favoured in reengineering applications,

because it tends to produce smaller and more tightly-linked clusters. Intuitively, in a cluster formed by the complete-link method, every member is similar to every other member. However, unlike single link and ward's methods that have known for their implementations in time O (n^2), the best complete link document clustering algorithm as far as known is O (n^2).

An entity could represent a subsystem, a directory, a file, a class, a function, a data structure, a variable, and so on. A resemblance coefficient for a given pair of entities indicates the degree of similarity or dissimilarity between these two entities, depending on the way in which the data is represented. A resemblance coefficient could be qualitative or quantitative. Whether qualitative or quantitative data should be chosen depending on the applications and attributes.

After applying clustering algorithm, the results should be evaluated and explained correctly. The clustering techniques adopted in the approach are based on numerical taxonomy. Numerical taxonomy uses numerical methods to classify entities. This method has conceptual and mathematical simplicity. The overall computational complexity of the algorithm is O (n^2) which is an "n x n" matrix. In fact, the computation runs quickly due to its simplicity.

5.6.2. Agglomerative Algorithms

The hierarchical agglomerative clustering approach can be described as follows. Given a set of documents:

1. Start with a set of singleton clusters, each containing one document and initially unmarked.

2. Repeat the following steps iteratively until there is only one cluster left unmarked.

a. Identify the two most similar unmarked clusters, and mark them.

b. Form a (new, unmarked) parent for these clusters by merging them together into a single cluster.

c. Updates the similarities between the clusters.

Agglomerative algorithms start at the bottom of the hierarchy: at the starting point there are N clusters that each cluster contains one entity (N is the number of entities). In each following step two clusters are joined. After N-1 steps all entities are contained in one cluster. Each level in the hierarchy defines a clustering. Now a cut point has to be determined. The clustering at the level of the cut point is the resulting clustering. Divisive clustering works the other way around. At the beginning all entities are contained in one cluster. In each step a cluster is split into two clusters. After N-1 steps, there are N clusters each containing one entity.

In hierarchical agglomerative clustering approach, each cluster in a given level clustering can be seen as a node in a hierarchy. When the cluster is a result of a merge, its direct descendants are its two sub-clusters in the previous level. The resulting hierarchy of a hierarchical method is a particular form of tree called a dendrogram, in which every internal node has exactly two children nodes. The entities in a dendrogram are represented by numbers in order of appearance from left to right. Cutting the hierarchy at the cut point would result in a cluster containing the two leftmost entities and a cluster containing only the rightmost entity.

5.6.3. Apply Agglomerative Hierarchical Clustering Analysis

For software decomposition purposes, the proposed approach chooses algorithms which impose a structure which satisfying the constraints that a good modularisation or componentisation should obey. An improved agglomerative hierarchical clustering

method is proposed to extract independent services from legacy code. The resulting hierarchical structure may be represented by a binary tree or "dendrogram", from which the desired clusters may be extracted. This clustering method analyses legacy code and expresses the results in a dendrogram which presents a hierarchic view of the legacy system. This hierarchic view consists of different levels of abstraction from source code to subsystems. This bottom-up clustering method is feasible and efficient for legacy system understanding.

Applying agglomerative hierarchic clustering analysis to identify components for using in Grid should follow the following rules:

Rule 1. Obtaining the data matrix.

For the legacy systems which are written by procedure-based code, the entities are defined as functions. In order to apply the clustering technique, the cluster technique is tailored based on function-function inter-connections. In this case, the data set represents the interdependencies or interconnections which are indicated by function calling relationships. The entity interconnections are defined differently regarding different software systems.

For objected-oriented programs, the coupling information could be class-attribute, class-method, method-method, the inheritance among classes, or a shared feature. In input data matrix, the 1 entries show the corresponding functions that are interconnected. The matrix is symmetrical and 1 entries are used in the main diagonal. The numeric values (1 or 0 entries), which show the number of interconnections among functions, are obtained from a static analysis of the source code.

The number of function calls based on syntax analysis, however it does not reflect the actual number of invocations of those functions. Dynamic information may be more insightful, but it is difficult to obtain and is highly dependent on run time behaviour.

Rule 2. Computing the resemble coefficients for the matrix.

To ascertain the similarity between two entities, the proportion of relevant matches between the two entities is calculated. There are different methods of counting relevant matches and there are many algorithms to calculate the similarity or resemblance coefficient. Method in this Grid oriented approach to calculate the similarity between functions is presented as follows, which is based on Sorenson coefficient [91].

Let a denote 1-1 match, which means the same attribute is coded as 1 for both entities. Similarly, let b, c, d denote 1-0, 0-1 and 0-0 match between two entities. Let Sxy be the resemblance coefficient for entities x and y, then the Sorenson coefficient: $Sxy = 2a / (2a + b + c)$. Compared to other coefficient definitions, Sorenson coefficient emphasises 1-1 match by giving twice weight. That is because the attributes are presented in both entities at the same time. Another reason to use Sorenson coefficient for software clustering is that it does not count 0-0 matches. From the software perspective, 0-0 matches represent software entities do not share commonalities and have no relations, so they should be ignored in the similarity function.

Rule 3. Executing the clustering algorithm.

In essence, the clustering algorithm is a sequence of operations that incrementally groups similar entities into clusters. The sequence begins with each entity in a separate cluster. At each step, the two clusters that are closest to each other (indicated by smallest Sorenson coefficient) are merged and the number of clusters is reduced by one. Once these two clusters have been merged, the resemblance coefficients between the newly formed cluster and the rest of the clusters are updated to reflect their closeness to the new cluster.

Instead of using single linkage (nearest neighbour) and complete linkage (furthest neighbour) rules as amalgamation rules, the proposed approach employs the UPGMA

method to find the average of the Sorenson coefficients when two clusters are merged in this clustering algorithm. In this method, the distance between two clusters is calculated as the average distance between all pairs of objects in the two different clusters. The input data is a collection of objects with their pair wise distances, and the output is a rooted tree (dendrogram).

This method is sometimes used for creating rooted phylogenetic trees under the assumption of a constant evolutionary rate. Initially, each object is in its own cluster. At each step, the nearest two clusters are combined into a higher-level cluster. The distance between any two clusters A and B is taken to be the average of all distances between pairs of objects in A and B. This method is also very efficient when the objects form natural distinct "clumps", however it performs equally well with elongated "chain" type clusters.

Then, dendrograms are adopted to demonstrate the clustering process and the proximity between entities. It presents a hierarchical view of the legacy system, which consists of different levels of abstraction from source code to subsystems. Figure 5.2 shows an example of dendrograms. The detail of its creation will be presented in the Chapter 9.

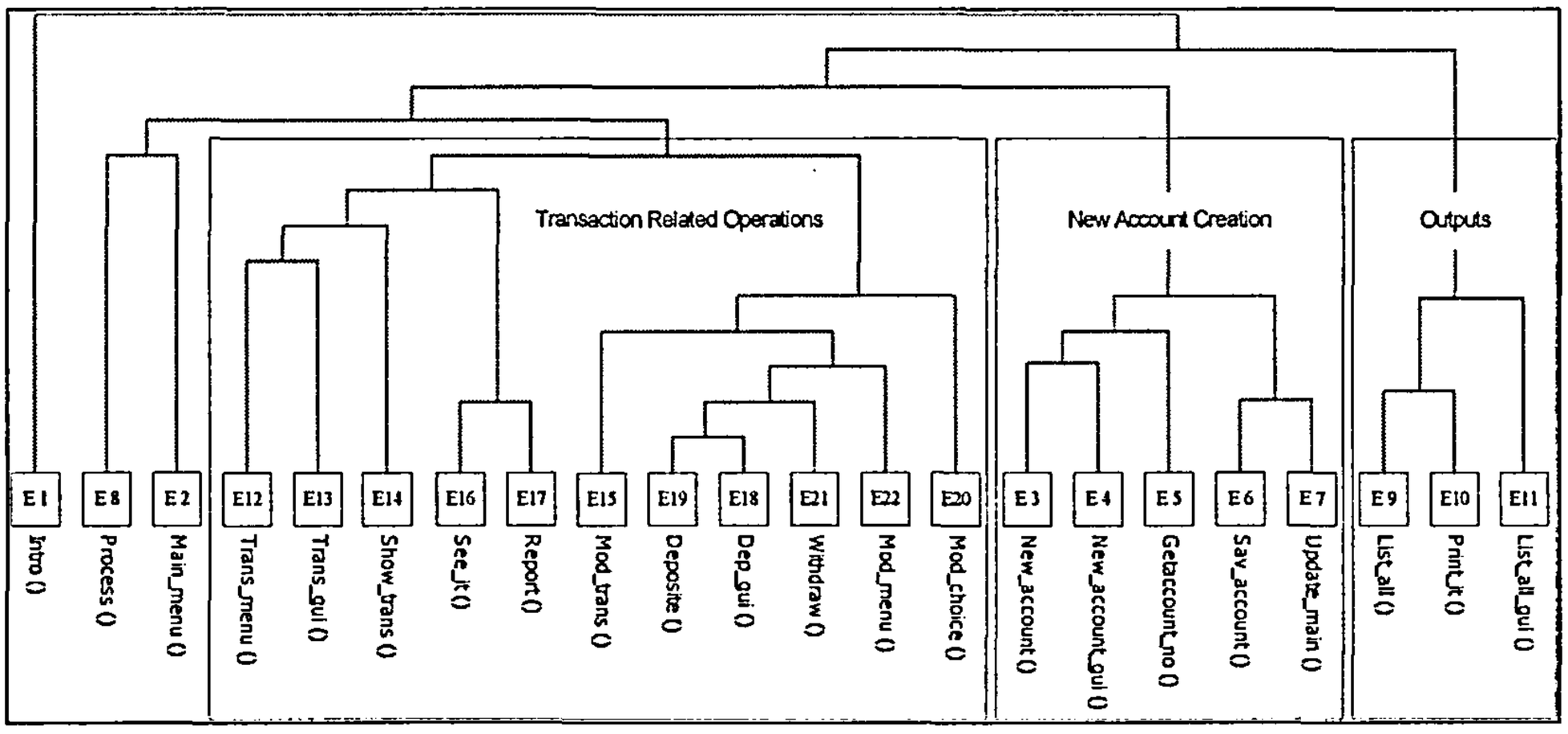

Figure 5.2. An Example of a Dendrogram.

In order to extract a functional component from legacy code, a cutting point must be figured out in the dendrogram. This cutting point affects the number and quality of the components derived from the dendrogram. It is an elementary factor for determining the granularity of extracted components. So far, the cutting point is determined with architects' participation and user requirements for the new Grid system. Other existed and recovered design information facilitates this decision-making process. This clustering method together with human supervision restructures legacy systems into components. The Gird component creation and packing will be presented in the following chapter.

5.7. Retargeting for Migration

Retargeting involves activities at the turning point between reverse engineering and forward engineering. If the three stages of reengineering (reverse engineering, functional restructuring, and forward engineering) is recalled, retargeting involves functional restructuring and the start of forward engineering.

In this research, Grid technology is often explained as a set of networking capabilities and software elements that support distributed resources sharing. The goal of Grid is to create virtual organisations to manage virtual computer of connected heterogeneous systems and distributed database for sharing various combinations of resources. A legacy system is an application program, which currently is a well-accepted and well-defined term within the software engineering community. Despite its poor competitiveness and compatibility with modern equivalents, it has been used until now since the cost of replacing or redesigning it is high. The implication is that the system is large, monolithic and difficult to modify.

Normally, the users' new requirements are added on top of the existing system when the system is reengineered, and these new requirements are implemented in a small number of program functions. This small number of functions can be ideally implemented using

reusable components from the reuse library. Since adding new requirements to the new system is carried out at the specification level, this stage is called functional restructuring.

In general, reverse engineering is the main thrust of reengineering because it addresses a very difficult problem: program comprehension. Forward engineering can be always undertaken with a sound and existing software development method. Nevertheless, in the case of the proposed reengineering approach, it is mainly concerned with integrating existing reusable components and newly developed components, and retargeting components integration in the Grid platform.

5.8. Summary

This chapter describes an approach to identify concerned resources from legacy systems for designing components which are used in Grid environment. The static program slicing techniques are applied to decompose program as a preliminary step for the migration of legacy systems. The control flow graph is adopted in the proposed approach for program representation and static analysis.

Then, software clustering techniques are applied to capture independent legacy code segments. An improved agglomerative hierarchical clustering method is proposed to extract independent services from legacy code. The resulting hierarchical structure may be represented by a binary tree or dendrogram, from which the desired clusters may be extracted. This clustering method analyses legacy code and expresses the results in a dendrogram which presents a hierarchic view of the legacy system. Reverse engineering techniques play an important role in this analysis process.

Based on the comprehension, legacy systems are extracted as concerned legacy code segments. These functional codes can be extended, isolated and fashioned as various Grid components which could be reused within different kinds of Grid systems.

Because the complete recovery of legacy systems is difficult to archive and the cost of reengineering the whole legacy system is expensive, reusing recovered legacy components is an economic and efficient way to reengineer legacy systems. The adopted software program slicing and clustering techniques are not new, but the comprehensive analysis ascertains the reusable legacy code efficiently. This chapter focuses on how to identify and retrieve useful resources from the legacy systems. The Grid components migration and packing will be presented in the following chapter.

Chapter 6

Grid Component Migration and Packing

6.1. Grid Middleware

Net-Centric Computing (NCC) is a distributed environment where applications and data are downloaded from servers and exchanged with peers across a network. It attracts significant interest from network and telecommunications practitioners as a vehicle for innovation.

As an emerging trend in NCC technology, Grid computing enables users to collaborate securely by sharing processing, applications, and data across systems to facilitate collaboration, faster application execution, and easier access to data. Grid focuses on large-scale resource sharing, innovative applications and high performance orientation. The sharing relationships may be static and long-lived or highly dynamic.

With increasing adoption of distributed systems and Web based applications, more and more existing applications turn to legacy systems. Grid technology can be applied to achieve enterprise-wide applications integration. Adapting Grid to evolve legacy systems could provide a rich set of capabilities. It allows organisations to use numerous computers to solve problems by sharing computing resources.

As these organisations vary tremendously in their purpose, scope, size, duration, and structure; particularly, the resource configurations of organisations have the potential to change dramatically. Providing Grid middleware for deploying existing applications is very useful to enable dynamic load distribution, fault resilience, ease system administration and data access locality.

As an interface of applications, Grid middleware may have following performance:

- Track which node has computational servers running and which one are provisioned with.
- Track each node's workload to locate the best choice for a given job request.
- Take care of the details in finding machines on which to execute computational tasks.

Reuse of existing protocols and seamless interfacing to existing applications is a high priority in the development of the Grid systems. The Grid middleware may allow legacy and new applications to operate seamlessly over Grid environment.

In software evolution area, Grid middleware technology provides an approach to develop high-performance, scalable middleware foundation based on Grid technology for legacy system evolution. Such middleware will provide transparent access to legacy resources concerning with process, achieve continuous computation and make better utilisation of the available computing resources among these organisations.

In addition, it investigates the architecture of large, distributed, computing infrastructures, high-speed networks to support legacy systems, and methodologies for effective software evolution on the underlying distributed infrastructure.

A Grid provides an abstraction for resources sharing and collaboration across multiple administrative domains. Resources are physical (hardware), informational (data) and capabilities (software). From the software evolution perspective, legacy resources are encapsulated into objects which can be used by the developers as components for their applications. Grid middleware can be acted as software that facilitates writing these applications and manages the underlying Grid infrastructure.

6.2. Process Transfer

With years of efforts, Grid researchers have successfully developed Grid technologies including security solutions, resource management protocols, information query protocols, and data management services. However, as the ultimate goal of Grid Computing is to design an infrastructure which supports dynamic, cross-organisational resource sharing, it is necessary to design an infrastructure to support dynamic, cross-organisational resource sharing, which acts as a solution for efficient and transparent task re-scheduling and management in the Grid.

To achieve continuous computation and to make better utilisation of the available computing resources among these organisations, runtime process rescheduling is required. Process migration is an attractive feature for re-scheduling tasks in Grid environments. It is the act of transferring a live process from one node to another node in the distributed system. It has been shown that process migration is very useful to enable dynamic to load distribution, fault resilience, ease system administration and data access locality. With increasing deployment of distributed systems in general, and distributed operating systems in particular, process migration is receiving more attention in both research and product development. One area where there is a need to understand the use of process migration is Grid Computing. With the shift from supercomputers to networks of workstations, and with the ever-increasing role of the Internet for high performance computing, process migration can play a more important role.

With the support of process migration, various runtime load balancing schemes can be employed for improving the execution efficiency of Grid applications. Process migration can also help those long-running applications by relocating them at suitable times to prevent interruption due to system activities or the execution of other applications. It also can help to relocate processes closer to the Grid point with data that they need to access.

Process migration consists of extracting the states of the migrating process on the source node and transferring it to the destination node where a new instance of the process is created. All the communication channels that concerned with the process are updated.

Benefits of Process Migration include:

- Accessing more processing power.
- Exploitation of resource locality.
- Resource sharing within a specific hardware device, large amount of free memory or some other resources.
- Improving Fault resilience by migration from a partially failed node, or in the case of long-running applications when failures of different kinds are probable.
- System administration is simplified if long-running computations can be temporarily transferred to other machines.
- Mobile computing also increases the demand for migration.

The types of application that would benefit from process migration include parallel applications, long-running applications, generic multi-user workloads, an individual generic application, migration-aware applications and network applications.

Various types of process migration implementations are possible. They include programming language support, object migration at the middleware level, system level process migration.

System level process migration is the traditional form of process migration, in which the researchers are mostly interested. The algorithm is developed with total kernel support using the kernel API. The module is implemented to work in privileged mode with accessing to all memory addresses.

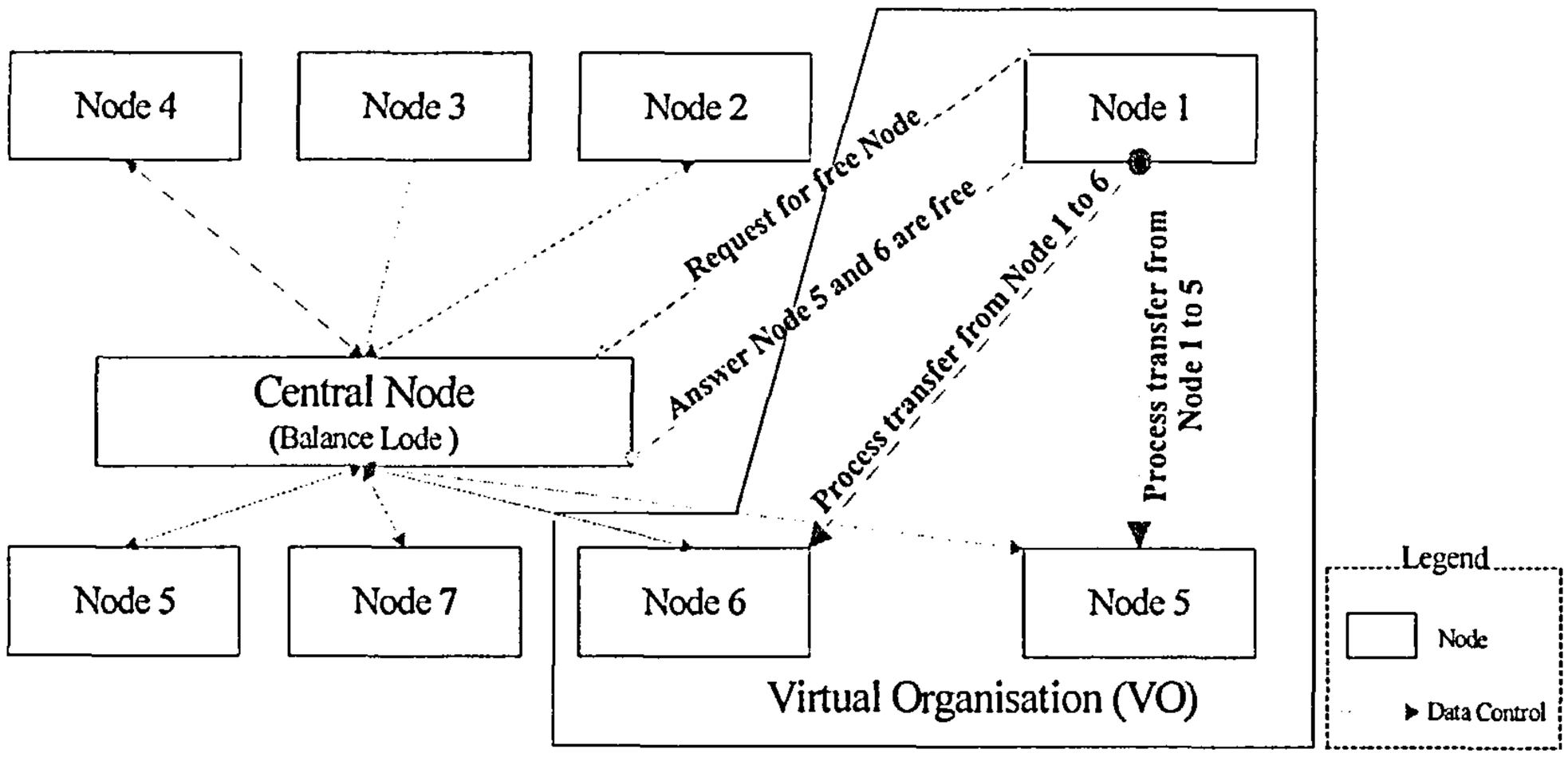

Figure 6.1. Framework of Process Transfer in Grid Environment.

Figure 6.1 shows the framework of process transfer in Grid Environment with N nodes. In this framework, there is one central node that is responsible for balancing load between the participating nodes in the Grid. All nodes can request the central node for some free node which is ideal or has fewer loads at any time. Then, the original nodes transfer some parts of its processes to other free nodes. These nodes performed as a Virtual Organisation to handle the work together.

The central node contains load-balancing algorithm to balance the loads of the participating nodes. It can use various statistics collected by this algorithm to find out the ideal or least loaded node. Then it can inform the requesting node with the details of the ideal node. The requesting node can communicate with the ideal node now and start the process migration.

As Figure 6.1 shows, node 1 has too much works, so even it could not handle itself. It requests the central node for some free nodes to help to do the work. Then, the central node answers that the node 5 and node 6 are free currently. Then, node 1, node 5 and node 6 constitute a virtual organisation to achieve the work together. In this VO, the

process can transfer form node 1 to node 5 and node 6, but also can transfer back from node 5 and node 6 to node 1, as well as transfer between the node 5 and node 6. This depends on the fewer node form time to time of the node, and this could be done by another Grid component named monitoring and discovery system (MDS).

Extending the three node system in the example to the whole Grid environment, legacy system could be restructured and reused in Grid systems and they will contribute a lot with these benefits.

6.3. XML for Information Exchange

The Extensible Markup Language (XML) is a broadly adopted format for structured documents and data on the Web. XML is a simple and flexible text format derived from Standard Generalised Mark-up Language (SGML) and developed by the World Wide Web Consortium (W3C). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible for HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML, and it is expanding from its original form in document processing and becoming a solution for data integration.

XML excels in inter-application data exchange because of its flexible and extensible method of describing data and its capability to communicate over the Internet using the standard HTTP protocol. XML supports tag extensions that allow various parts of Web based applications to exchange information. The popularity of using XML as exchange format comes from its inherent advantages such as separating content and representation and implementing validation and stylesheets in XML too. The most important advantage of it is, however, that XML is the de facto standard for information exchange and has been adopted as the foundation for data exchange by almost all software products

published recently. The XML role of exchanging information of XML is a significant part of transforming a legacy system into an evolvable one.

For example, the J2EE and B2B are two popular platforms for evolving a legacy system that could meet the requirements of an evolvable system. The B2B architecture supports distributed business objects loosely coupled with XML messages, while the J2EE architecture supports components that encapsulate business logic and reside in a container providing the runtime environment and other services such as transactional support.

In order to make a distributed system that has loosely coupled components and provide enterprise functionality, it makes sense to combine the two architectures by decomposing each business object using the J2EE architecture and connecting business objects using the XML messaging system of the B2B architecture.

This flexibility makes XML a powerful mechanism for B2B application integration. B2B integration is the automated exchange of information between systems from different organisations. XML-based B2B is gaining momentum as XML vocabularies emerge in specific business domains such as finance. In addition, a growing number of commercial enterprise application solutions are embracing XML.

6.4. Integration of Legacy Resources and Grid with XML

XML has been heavily used throughout modern software development. Its application has almost become compulsory in software partitions and is increasingly becoming the mandatory for a genuine evolvable system. A software system built couples of years ago seems undoubtedly a legacy system. Most legacy system were developed using procedural language such as C, COBOL and Pascal, while most modern system are being written in object oriented languages such as Java, C++ or C#. Using system duration or implementing

language as metric is therefore insufficient for defining a legacy system unambiguously. For the purposes of this discussion, XML is introduced to describe and manipulate legacy systems for Grid evolution.

Grid computing architecture supports distributed object loosely with XML messages and encapsulated components, which could be resided in a container providing the runtime environment and other services such as transactional support. The core mechanism of the B2B architecture is an XML messaging system containing a set of business processes. The XML content describes a data dictionary description of the elements that make up the XML content, and a messaging service that specifies how the XML content is packaged and transferred.

Grid service is an implementation of service based pattern like Web services. It builds a distributed programming model based on XML standards that a service consumer initiates an operation of a service specified in XML via XML-based SOAP standard. XML is the key technology that enables a large number of participants to corporate in a distributed programming environment.

Grid is one of the most important technologies for enterprise legacy systems integration, which could be adopted in three ways depending on the exposed services. Although the strategies vary with different legacy data integration, they are all based on the same principle. XML scheme are employed to describe the legacy data, and their message transactions are represented by XML based on SOAP. Once legacy data is in an XML format, its manipulation becomes much less daunting, and its development also becomes easier.

With XML technologie, Grid allows legacy and new applications to be interoperated. And it can create composite applications by combining interfaces to individual applications with various data sources using XML as the standard data format. XML in

the proposed solution to Web based systems evolution has two roles: model documentation and systems integration.

6.5. Component Representation Using XML

Both bottom up and top down approach help to extract and represent legacy assets into XML format. The bottom-up approach utilises the concept of Grid component definition that denotes the syntactic structures of a programming language. The top-down approach examines the grammar of the specific programming language, and defines a standard logical structure.

In the proposed approach, a structure of the Abstract Syntax Tree (AST) has been defined to develop Grid components. By recursively traversing the hierarchy of the component entities, it is able to map the Grid component to a Document Type Definitions (DTD). Specifically, the tree hierarchical structure of the Grid component is mapped into XML elements and attributes. Each node and edge in the AST is mapped to an XML element tag and the attribute values of an AST node are mapped to the corresponding attribute values of the XML elements.

The Grid component and its corresponding DTD can be enhanced with information. Similarly, Grid component generalisations include the introduction of elements that relate to system constructs. In this context, the grammar of the programming language being modelled defines the DTD and consequently the organisation of the XML document that models the AST of a given source code fragment.

XML is increasingly been accepted not only as a standard for describing documents, but also as a data description language for all types of information. DTDs are generated to describe the characteristics of the data and to make the documents self-contained and

usable as a data exchange format. Combined with client-side interpreted XSLT stylesheets, it provides an approach for publishing data from relational databases on the Web.

6.5.1. AST Representation

In computer science, an AST is a finite, labeled, directed tree, where the internal nodes are labelled by operators, and the leaf nodes represent the operands of the node operators. Thus, the leaves have null operators, such as variables and constants. In computing, AST is used in a parser as an intermediate between a parse tree and a data structure. The latter is often used as a compiler or interpreter's internal representation of a computer program while it is being optimised and used to perform code generation. The range of all such possible structures is described by the abstract syntax. An AST differs from a parse tree by omitting nodes and edges according to syntax rules which do not affect the semantics of the program. The classic example of such an omission is the grouping parentheses, since in an AST the grouping of operands is explicated in the tree structure.

Creating an AST in a parser with a language described by a context free grammar is straightforward. Most rules in the grammar create a new node with the node edges being the symbols in the rule. The rules that do not contribute to the AST, such as grouping rules, merely pass through the node for one of their symbols. Alternatively, a parser can create a full parse tree, and a post-pass over the parse tree can convert it to an AST by removing the nodes and edges not used in the abstract syntax.

AST have been successfully used by the data flow analysis and compilers community for the purpose of analysing and transforming source code entities. Such tree-like structures represent the source program in a top-down matter. The internal nodes of the AST represent the non-terminal phrases of the program text, such as statements, operations, and functions. The leaf nodes represent terminal symbols, such as identifiers, and type

declarators. An edge between nodes in the AST denotes attributes which are modelled as mappings between AST nodes.

Interface creation is processed by a data flow analysis which identifies all of the variables directly referenced by the function. This includes overlaying data structures and indexed arrays. If an elementary item in a structure is referenced, then the whole structure is included in the interface.

Similar to AST, an XML document can be thought of as a tree structure with nodes and edges connected by a hierarchy relationship. Compared to the custom-made software extractors, this approach provides a standard API for programmatic access to XML documents, and allows for the manipulation of structured data. In such a way, tool developers can consistently interact, exchange and transform XML documents that correspond to source code representation and analysis results. In addition, they are widely supported and can be used with different programming languages, such as C++, Java, and JavaScript.

Finally, XML documents can be easily transported by the HTTP protocol over the Internet. Therefore, XML allows an AST information base and analysis tools to be independent of the parser, and published in a Web-based repository.

The software analysis tools can either download the required information over the network or store the AST locally for further processing. Results obtained from the analysis of large systems can also be represented as annotations of the AST in an XML form, and directly stored in a Web repository.

For the circumstances of Web based legacy systems evolution, Web systems often evolve from small and simple collections of purely HTML pages which defines the style, structure, and content of the Web pages to complex applications, offering advanced transactions and data access. They also can be seen as software systems comprising thousands of line of "code" (HTML) split into many "modules". Static HTML pages are

programs with encapsulated data, which infringe software engineering good practice. As HTML is a short-constrained language, it is useful to check the validity of these sources before working on it [90].

The analysis of Web pages usually starts at the specified root URL. Firstly, it determines the type of the URL. If The URL is a HTML pages, it extracts its entire links to other Web page, and subsequently analyses them in the same manner as the root URL. Given the description of each Website page in terms of feature vectors, it is possible to exploit similarity or distance measures to agglomerate entities into clusters. Web pages can easily be moved without affecting the links if the file structures are preserved.

Mostly, Websites are built with pure HTML pages, which have some embedded contents such as images and increasingly the inclusion of scripts both as part of the HTML document or as a means of generating the HTML page. And the designers usually highlight interesting output information with HTML tags, such as <b> or <tr> or <td>, or with a distinct highlight font attribute, such as colour. Also, the file path is a good clue to detect the page types through a Website.

To enable the extraction of data and structure, first, AST has been built to identify the corresponding paths and possibly some properties of the elements themselves, then, strings which are represented by the text of content leaves to every node of the tree are associated as the attributed elements.

In order to extract instances from HTML document, rules must be formulated for how to traverse the tree structure of the document in order to locate the instances of the constituents. These rules are formulated using XPATH in order to localise the HTML tree that spans all the concept constituents.

Now, employing the cluster and slicing techniques, the HTML documents could be clustered and cleaned and the contents are separated from layout by integrating in HTML pages scripts for retrieving the dynamic data from a database.

After parsing, the system stores a rationalised copy of the extracted HTML data in central data storage with each unique element stored in a separate part of the repository. A relational database can be used for the central data storage. Rather than storing the details of how to reconstruct the pages, the original HTML page is modified to use scripts to retrieve the data from storage and regenerate pages as required. These modified files only contain the structure of the original HTML files. The files are stored in an organised directory structure on the server.

Figure 6.2 shows an example of an abstract syntax tree. The detail of its creation will be presented in the Chapter 9.

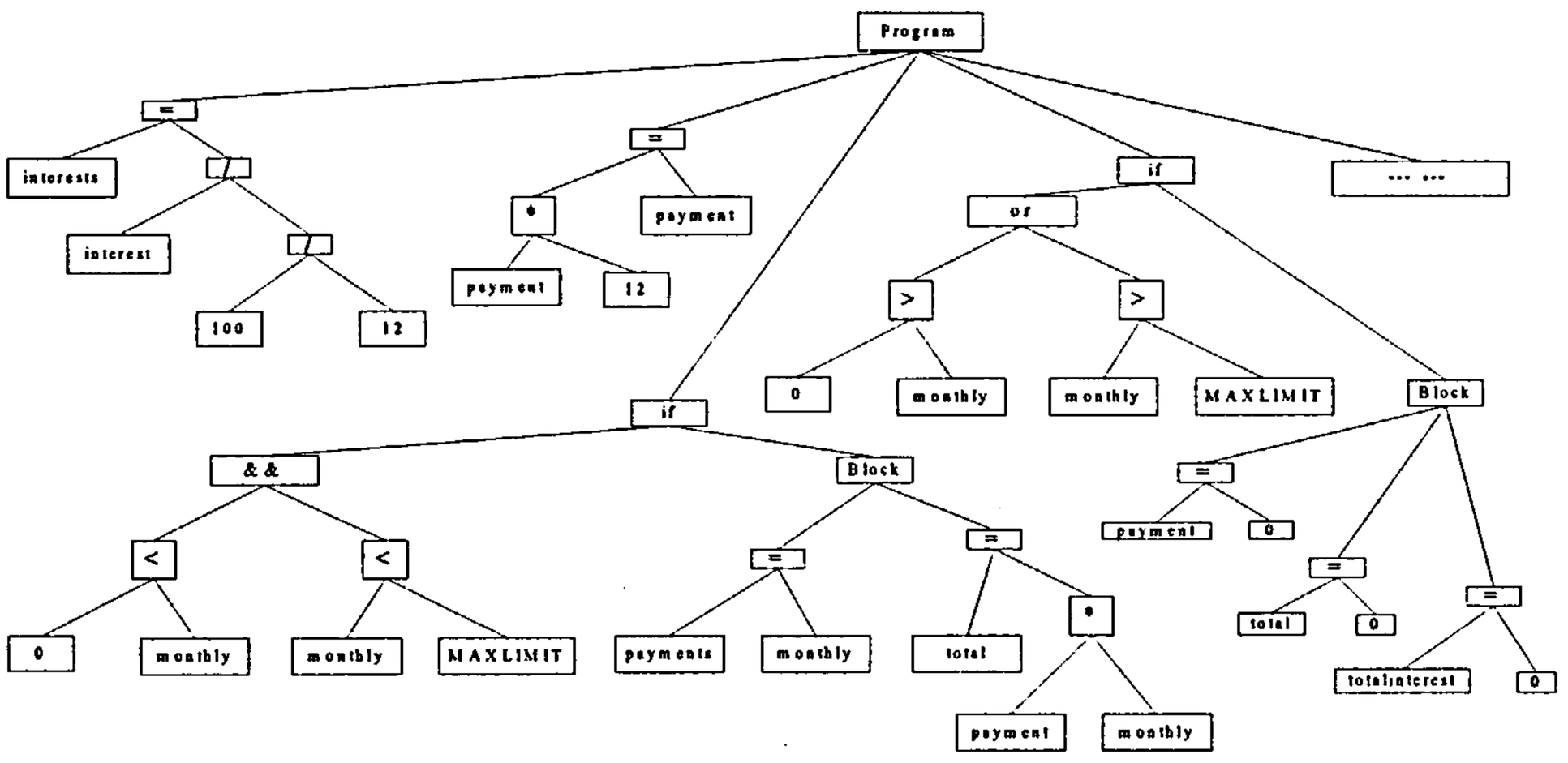

Figure 6.2. An Example of an Abstract Syntax Tree.

6.5.2. DTD Expression

DTD defined slightly differently by the XML and SGML specifications, is the term used to describe a document or portion thereof that is authored in the DTD language. It is also a holdout from XML legacy as a special case of SGML. They were intended to make certain that an application could read an SGML file by knowing what to expect in terms

of the document's structure. It can be contained in an external file or as a part of the XML document itself.

In this thesis, the term "Grid component" generally refers to a collection of reusable legacy resources. Typically, it describes the relation of legacy resources and the semantic rules governing the resources.

For a programming language, Grid component specifies a set of nodes and arcs in an abstract syntax tree and denotes the structure of the elements in the language. The nodes denote language constructs such as the declarations, functions, and statements. The arcs specify the attributes of the nodes and associate the parent and the child elements. In addition to the information obtained from the parsing tree, Grid components can be enhanced by the annotations that are not belonged to the syntactic parsing tree, for instance, the node identifier, usage, and linkage. This information is represented by additional attributes of a node. In order to represent the concerned legacy resources for Grid component interchange and integration, the approach of encoding a Grid component in DTD and effectively validate XML based source representation is used.

A DTD is primarily used to express a schema via a set of declarations that conform to a particular markup syntax and that describe a class, or type, of SGML or XML documents, in terms of constraints on the structure of those documents. A DTD may also declare constructs that are not always required to establish document structure, which however may affect the interpretation of some documents.

As an expression of a schema, a DTD specifies, in effect, the syntax of an "application" of SGML or XML, such as the derivative language HTML or XHTML. This syntax usually is not as general as SGML or XML syntax. In a DTD, the structure of a class of documents is described via element and attribute-list declarations. Element declarations name the allowable set of elements within the document, and specify how declared element of character data are contained within each element. Attribute-list declarations

name the allowable set of attributes for each declared element, including the type of each attribute value, if not an explicit set of valid value.

It is possible to map a specific language construct into a DTD specification when a specific programming language and its corresponding grammar are given. To accomplish this mapping, a set of transformations is defined to convey the grammar of the programming language to the DTD.

Taking the advantage of concepts in object modelling, general-purpose transformation rules can be used to enable DTD declarations conversion. The DTD production rules in the Grid orientd legacy system evolution approach are specified as follows:

1. Each class is mapped into an element of the DTD declarations.

2. Each attribute with a primitive type is mapped into an attribute of the DTD element.

3. Each attribute with a user defined type (i.e. class) is mapped into the DTD element that is contained in the element corresponding to its containing class.

4. Each aggregation is ignored.

5. Each multiplicity constraint for an attribute association is mapped into the DTD quantity control.

6. The attribute association is ignored since it is treated as equivalent to its attribute.

The XML DTD establishes a formal set of rules to define the document structure. The relation between a DTD and its XML document is equivalent to the relation between Grid component and its instances. Therefore, a DTD specification can be used to express the structural aspects of a Grid component, where its instances are encoded in the XML document.

6.5.3. XSL Transformation for System Evolution

A Web application is an application that is accessed with a Web browser over a network such as the Internet or an intranet. Web applications are popular due to the ubiquity of the browser as a client, sometimes called a thin client. The ability to update and maintain Web applications without distributing and installing software on thousands potential of client computers is a key reason for their popularity. Web interfaces have been used increasingly for applications that have been thought of previously as traditional, single-user applications. In this thesis, Extensible Stylesheet Language (XSL) is introduced to achieve the Web interface transformation and to provide a better control over legacy systems.

Web application can be seen as software systems composed by thousands of line of source code (HTML). Static HTML pages may comprise programs and encapsulated data. However, they violate software engineering good practice sometimes. Extensible Stylesheet Language Transformations (XSLT) is an XML-based language which is used for the transformation of XML documents. It can manipulate XML from one structure into another. Therefore, it can be implemented for transforming XML into HTML or other XML document structures. An XSLT stylesheet can make use of information from a schema, and an XSLT transformation can take place in the absence of a DTD. The XSLT processor has access to the type information associated with individual nodes, not merely to the untyped text. The important roles of XSLT is to add styling information to an XML source document, by transforming it into a document consisting of XSL formatting objects or into another presentation-oriented format such as HTML, XHTML. However, XSLT is used for a wide range of transformation tasks, not exclusively for formatting and presentation applications.

In Grid oriented software evolution, the application of XML has not replace HTML documents completely. HTML is created for content presentation in client browser. Its

elements and attributes are dedicated to document formatting and user interactive actions such as collecting user input information from the browser and submitting it to a server. HTML should be used as a protocol for information representation and collection, but not for information storage.

While both HTML and XML are languages representing semi-structured data, HTML is mainly presentation oriented and not really suitable for database applications. Keeping information in HTML documents tightly couples the content and its representation, which makes it difficult for information reuse and maintenance. The weakness of HTML in information storage is overcome by the use of XML, which is ideal for keeping data. Many Web-based systems can dynamically produce client side HTML pages from XML documents via XSLT and is used to format or transform XML content from one form to another. The benefit of the approach is that without any efforts to duplicate the legacy code, all features from legacy systems are inherited and they can be enriched with Grid features. Furthermore, they can easily be integrated with each other to perform dynamic services in a Grid environment.

XSL can be used not only for formatting an incoming document structure but also for formatting the data returned from the legacy system into a usable structure for the function or application using this API. XSLT processing often begins by reading a serialised XML input document into the source tree and ends by writing the result tree to an output document. The output document may be XML, but can be HTML, plain text or any other format that the XSLT processor is capable of producing. The transformation is achieved by a set of template rules. A template rule associates a pattern, which matches nodes in the source document with a sequence constructor. The rules of XSLT transformation in the Grid oriented legacy system evolution approach may include:

1. Read the XSLT stylesheet with an XML parser and convert its content to a tree of nodes (the stylesheet tree), according to the XPath data model.

2. Read the input XML with an XML parser and convert its content to a tree of nodes (the source tree) according to the XPath data model.

3. Refine the stylesheet tree and the source tree.

4. Supplement the stylesheet tree with a trio of built-in template rules that provide default behaviours for any node type that might be encountered during processing.

5. Process the root node of the source tree and serialise the result tree.

When processing a node, the following steps are undertaken:

1. The best-matching template rule for the node is located. This is facilitated by each template rule's "match" pattern (an XPath-like expression), indicating the nodes to which it can be applied. Each template is assigned a relative priority and import precedence by the processor to help ease conflict resolution. The order of template rules in the stylesheet can also help to resolve conflicts between templates which match the same nodes, but it does not affect the order in which nodes are processed.

2. Template rule contents are instantiated. Elements in the XSLT namespace are treated as instructions and have special semantics that guide how they are interpreted. Some results in nodes are added to the result tree, and others are control oriented. Comments and processing instructions are ignored.

6.5.4. An Example

In order to demonstrate the previous rules, the proposed approach is applied to the De Montfort University Managed Learning Environment (MLE) [90]. MLE is a resources management and learning system for students working. It manages student course and timetable management. And students can read their personal information and public

announcements as well. Even more, it support interact student union voting and online student accommodation searching.

The benefits of reengineering MLE system into Grid services include making the MLE system work not only for students in De Montfort University, but also for consumers who want to use this service and paid for it. Also, the Grid service oriented evolution will make the MLE system that works as a stateful resource and integrated with other functional stateful resource into a new powerful service possible.

For the needs of this example, it focuses on examining a simple page. Figure 6.3 gives a screen-shot of the course management Web page on the MLE Web system. Concerning of the data structure, this page describes the module information and course title of a PhD student, as well as some other related information. It also contains a picture of DMU and many other related links.

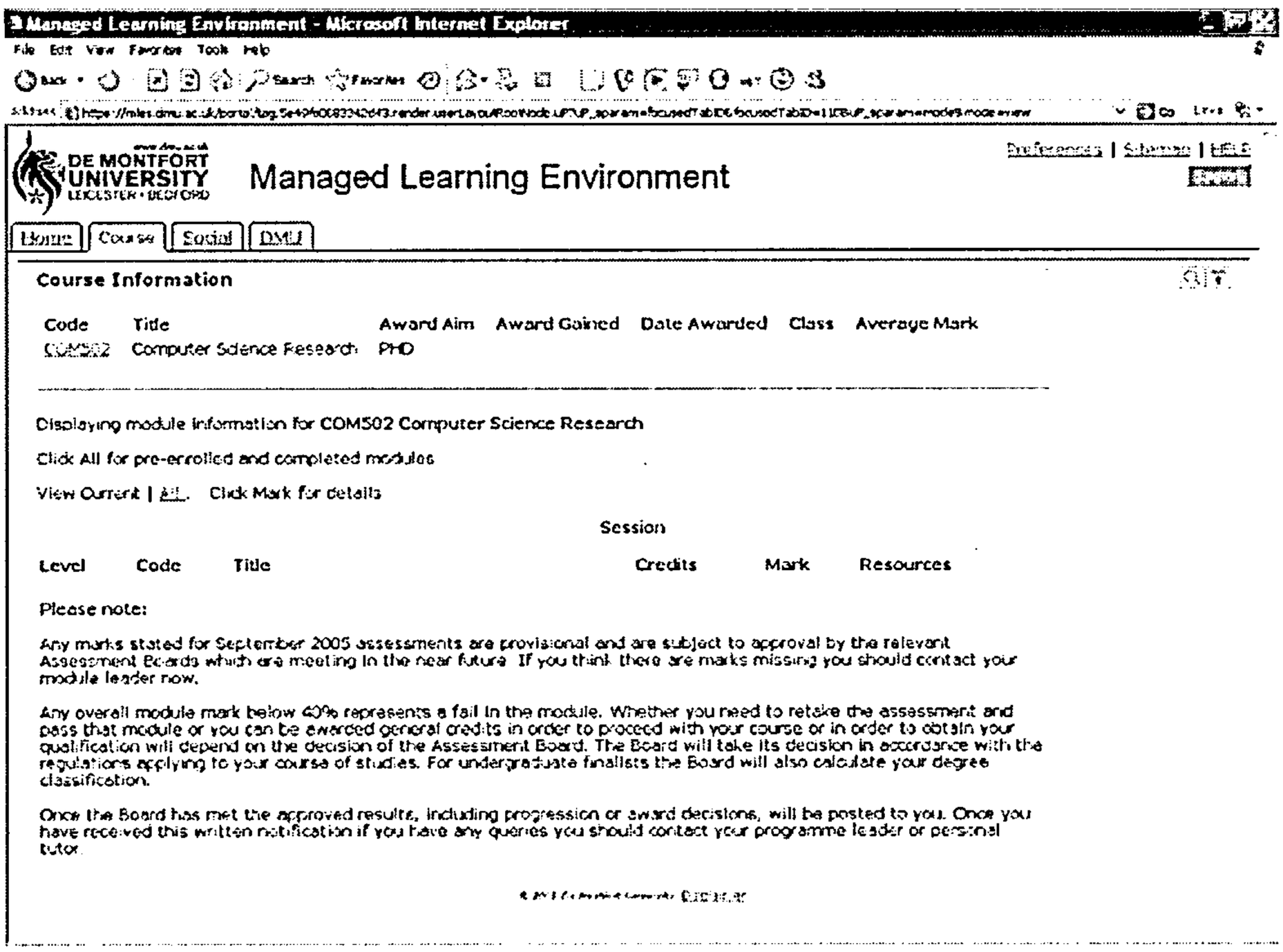

Figure 6.3. Screen-Shot at a Page of MLE System.

The process of representation relies on the use of XML scheme for creating patterns in hierarchical order. When facing a new HTML document, the only pattern is <documents> with a unique instance.

In the extraction step, the HTML document has been parsed into a tree representation rooted at the <html> tag, and the document subtrees possibly containing information of interest that can be efficiently located. Each node in the tree represents a pair of matching HTML tags enclosing a part of the document.

The label on the node corresponds to the entity of the target concept contained in this part of the document. Then, the HTML elements of the document parse tree can be easily marked by XML patterns. The output by the extractor is well-suited for translation. By exploiting the hierarchical structure of the pattern instance base and using pattern names as default XML element names, the HTML attributes can be translated and retained in the XML output.

```
<!ELEMENT Managed Learning Environment (Course*)>
<!ELEMENT Course (Description, Code, Title, Aim, Information?)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Code (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Aim (#PCDATA)>
<!ELEMENT Information (#PCDATA)>
```

Figure 6.4. DTD Representation for a Page of MLE.

Figure 6.4 shows the DTD representation of MLE. Elements in this example include:

1. Managed Learning Environment is a valid element name, and an instance of such an element contains any number of course elements. The "*" denotes there can be 0 or more course elements within the Managed Learning Environment element.

2. "Course" is a valid element name, and an instance of such an element contains one element named "Description", followed by "Code" element, then "Title", "Aim" and "Information" (optional). The "?" indicates that an element is optional. The reference to the "Description", "Code", "Title" and "Aim" elements name has no "?", so a person element must contain "Description", "Code", "Title" and "Aim" element.

3. "Description" is a valid element name, and an instance of such an element contains character data.

4. "Code" is a valid element name, and an instance of such an element contains character data.

5. "Title" is a valid element name, and an instance of such an element contains character data.

6. "Aim" is a valid element name, and an instance of such an element contains character data.

7. "Information" is a valid element name, and an instance of such an element contains character data.

```
<? Xml version= "1.0"  encoding=  " UTF – 8" ?>
<documents>
  <Website> Managed Learning Environment </Website>
  <WebsiteName> MLE </WebsiteName>
  <Entry>
      <Sort> Course </Sort>
          <Description>  Course Information</Description>
           <Code>  COM502  </Code>
           <Title>  Computer Sciences Research  <Title>
           <Aim> PHD  </Aim>
           <Information>  Please note: Any marks stated for  ...   </Information>
      <Picture/>
  </Entry>
[…]
```

Figure 6.5. XML Representation of MLE Web Page.

Figure 6.5 shows the XML representation which makes use of and DTD conforms to it. By analysing the MLE Web systems, a sort of patterns could be created, such as: "Website", "WebsiteName", "Sort", "Code", "Title", "Aim", "Description", "Information", and so on.

It is possible to render this in an XML enabled browser by pasting and saving not only the DTD component above to a text file, but also the XML file to a differently-named text file, and opening the XML file with the browser. All the files should be saved in the same directory. However, many browsers do not check that an XML document conforms to the rules in the DTD; they are only required to check if the DTD is syntactically correct. For security reasons, they may also choose not to read the external DTD.

By applying the XSLT transform in Figure 6.6, the XML representation of MLE can be transformed to a new XML document which has another structure as shown in Figure 6.7.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
 <xsl:template match="/">
   <transform>
     <xsl:apply-templates/>
   </transform>
 </xsl:template>
 <xsl:template match="Course">
    <record>
      <Code>
        <xsl:value-of select="@Code" />
      </Code>
      <Title>
        <xsl:value-of select="@Title" />
      </Title>
      <Aim>
        <xsl:value-of select="@Aim" />
      </Aim>
    </record>
  </xsl:template>
</xsl:stylesheet>
```

Figure 6.6. XSLT transformation of MLE.

```
<?xml version="1.0" encoding="UTF-8"?>
<transform>
  <record>
    <Code> COM520 </Code>
    <Title>Computer Sciences Research</Title>
    <Aim>PHD</Aim>
  </record>
</transform>
```

Figure 6.7. New XML Representation of MLE Web Page.

In this example, the HTML presented content information of the MLE system could be transformed to XML represented document based on AST representation, DTD transformation, XSLT stylesheets transformation, XML Schema definition of content model, and the XML file containing the page content itself.

The generated XML documents are usually larger than the source code in size, because of all the tags and attributes that are added to the source code. They can be considered as a dynamic database which can be easily manipulated. Similarly, the extracted information can be stored in commercial relational databases by storing XML documents as one file in a table, or multiplying fields in one or more tables. All in all, by following different rules, necessary information can be extracted from the source code and they can be encoded in a uniformed and application-independent format.

6.6. Wrap as XML Component

Interaction compatibility is a main issue to integrate legacy code in Grid environment. Legacy system in enterprise has a variety of data formats and data semantics. Different company use different language to describe data, so the main difficulty of integration legacy assets is interaction compatibility. To use various legacy systems in one Grid environment, they mush supply a common interface to share rescues and communication each other.

The proposed approach chose Java as common language to evolve legacy systems as it has many benefits such as it is an object orientation language, and it is a platform independence language and it has automatic garbage collection mechanism. In the proposed approach XML is used to describe the structure of the data, XSL language is used to manipulating XML from one structure into another. And Java is used to encapsulate them with a few simple classes and implement the application.

For the purpose of importing existing XML represented information sources to Grid services domain, the interfaces are essentially defined in a complex XML type. It is attributes to the name, type and name space. The XML grammar can be used to identify all publicly available methods, their signatures, and their return types.

The XML component is considered as dynamic resources that can be easily manipulated by Grid applications. XML component is an XML schema which is used to describe component meta-data and act as a common intermediate language between the elements of the component framework [92]. In addition, it provides a means by which the diverse units of the framework may communicate, and is used to describe the meta-data for the abstract components, the component implementations, resources and applications. It is a suitable target language for a number of possible end-user tools, without imposing constraints on their configuration.

All components must have at least one implementation. The component specification in XML Component is placed in the component repository, together with meta-data that describes its behaviour and interface. Each component implementation corresponds to a particular component specification. New component implementations are placed within the repository, along with meta-data describing their performance characteristics and resource requirements. The implementation meta-data is a XML Component document, which is distinct from but linked to the component specification.

The application builder produces a XML Component application description document. This document consists of the <network> and <repository> information. The network represents a composition of component instances together with any user users. The composition is a simple typed port and network mechanism, consisting of <instance> elements together with <dataflow> connectors. The connectors are attached between sources and sink ports according to the component type. The user may customise the component by specifying simple values that are recorded as <property> elements.

The choice of components within the builder tool is determined by those elements contained within the repository. The repository XML Component data provides the interface information for the <component> types, specifying <port> and <property> elements. The types and default properties specified in the repository XML Component allow customisation where it is required.

The repository component information also indicates component inheritance and package information. The repository also contains information needed to create the run time representation, namely the meta-data for the methods in the component implementation. These are represented by <implementation> and <action> elements. The <action> elements specify the bindings to ports and the location of corresponding performance data. The package and location information for the executables are stored alongside the implementation data in the repository. These are referred to <object> elements. The example of XML component creation are presented in Chapter 9.

6.7. Summary

After the components identification from legacy systems, they have to be migrated for deploying in the Grid environment. This chapter presents the approach of migration and packing the extracted legacy assets as Grid components.

The proposed approach is based on XML representation and transformation. Once a software component has been extracted from a legacy system, or has been built as a new component, its interface can be extracted and represented in XML. In this thesis, the XML representation is not only finished by the component wrap, but also with the sources code analysis included such as the using of AST, DTD and XSLT. At last, the legacy components are wrapped as XML components which could be further used in the Grid services environment.

Chapter 7

Grid Service Integration

A Grid can be defined as a layer of networked services that allow users single sign-on access to a collection of distributed computing, data and application resources. The Grid services allow the entire collection to be seen as a seamless information processing system that the user can access from any location. It is basically the Web services with improved characteristics and services.

Grid services have emerged by combining Web services and Grid computing to perform a seamless information processing system across distributed, heterogeneous, dynamic virtual organisations. Web services are the technology for Internet based applications with loosely coupled clients and servers. However, the implementations of Web services are typically stateless. Compared to this, Grid services have evolved to make it possible to dynamically share and coordinate heterogeneous service resources.

The Grid service oriented reengineering could bring great benefits for both legacy systems and Grid systems. It performs as reusing useful legacy system resources into the Grid service environment. Comparing with design a new system, the proposed approach is less risky and highly transparent, and it is massively reducing time and cost.

This chapter focuses on addressing the following issues, including:

- Design of a Grid service enabled integration architecture that allows the legacy systems to easily interact and inter-operate with other Grid service.
- Specification of the behaviour of identified components in terms of their well defined interfaces.

- Definition of appropriate middleware to integrate the identified components to a heavily heterogeneous Grid service environment.

7.1. From Web Service to Grid Services

Grid services are basically Web services with improved characteristics and services. The most important improvement is that Grid services are stateful and transient services. It can remember what have been done from one invocation to another with the support of stateful resources.

As Grid services are dynamic and stateful, a new mechanism is needed to address this issue. The OGSI specification version 1.0, released in July 2003, defines a set of conventions and extensions for the use of WSDL and XML schema to enable stateful Web services. However, there are criticisms of OGSI from the Web service community. For instance, OGSI has too much stuff in one specification. It does not work well with existing Web services and XML tools. Also, it is too object oriented, and it does not support the forthcoming WSDL 2.0 standards.

As the initial standard, OGSI have the following disadvantages:

1. Too much stuff in one specification. OGSI dose not have a clean separation of functions to support incremental adoption.

2. Does not work well with existing Web services and XML tooling. OGSI v1.0 uses XML schema aggressively, for example with substantial use of xsd: any, attributes, etc., and "document-oriented" WSDL operations. These features cause problems with, for example, JAX-RPC. WSRF uses standard XML Schema mechanisms that are familiar to developers and are supported by existing tooling.

3. Too object oriented. OGSI v1.0 models a stateful resource as a Web service that encapsulates the resource's state, with the identity and lifecycle of the service and

resource state coupled. This approach has spurred anxiety among some Web services purists

4. It does not support the forthcoming WSDL 2.0 standards. The OGSI authors exploited constructs from the proposed WSDL 2.0 draft specification. Delays in the publication of WSDL 2.0 made it more difficult to support the OGSI definition with existing Web services tooling and runtimes.

As a substitution, WSRF, as well as the related WS-notification family of specifications has been proposed. WSRF and WS-notification capture all of the functionality provided by OGSI, but this is done by integrating better with evolving Web service standards.

Specifically, the WSRF definition relies upon the WS-addressing specification. It is a new way for manipulating "stateful resources" to perform Grid services. It defines the concept of "stateful resources" and how they can be discovered, queried and manipulated via Web services. In addition, the WSRF definition expresses the capabilities of the OGSI definition in a way that is more consistent and will be more familiar to Web service developers.

The WSRF has been used to define conventions for modelling and managing state in distributed systems based on Web service context, and WS-Notification. It consists of a set of specifications: WS-Resource Properties, WS Resource Lifetime, WS-Base Faults and WS Service Group, which define how WS-Resources are named, discovered, queried, indexed, altered, and how their lifetimes are managed.

The changes from OGSI to WSRF are primarily syntactic, but they are also represented some useful progress. The separation of OGSI functionality into six independent specifications simplifies adoption. The use of WS-Addressing is a step forward, and less aggressive use of XML Schema and WSDL 2.0 features will facilitate the use of available tooling.

The definition of the WS-Resource framework facilitates the construction and use of interoperable services, by making it possible for different service providers and service consumers to describe, access, and manage their stateful resources in standard ways. Equally importantly, the framework introduces support for stateful resources without compromising the ability to implement Web services as stateless message processors. The framework also addresses issues of renewable references, grouping, notification, and fault reporting.

A Grid service has an identity, service data, and lifetime management mechanisms. A WS-Resource has a name, resource properties, and lifetime management mechanisms. Although the terms have changed, but the need, concepts, or mechanisms are same. Both OGSI and WSRF provide the mechanisms required for the really important specifications, that define the overarching Open Grid Services Architecture (OGSA). Working on OGSA is continuing, and it is affected only slightly by these changes.

The WSRF proposal is a refactoring of OGSI concepts to align with Web services better. And the current technical ground swell appears to be that WSRF is a better starting point with respect to obtaining wider community acceptance as well as rapid developments of usable tools and Grid applications.

7.2. An Architecture for Grid Services Integration

7.2.1. Stateful Resources

Being stateful is a major feature in Grid systems. With the support of stateful resources Grid can remember what have been done from one invocation to another. The term state is vague and can encompass many different aspects of a computer system, from the value stored in a specific database record to the seek time or even temperature of the disk drive.

In this thesis, a stateful resource can be defined as resource which have following features:

- have a specific set of state data expressible as an XML document.
- have a well-defined lifecycle.
- to be known by one or more services.

Examples of system components that may be modelled as stateful resources are the files in a file system, rows in a relational database, and encapsulated objects. A stateful resource can also be a collection or group of other stateful resources. Its state may be implemented as an actual XML document that is stored in memory, in the file system, in a database, or in some XML repository.

Alternatively, the same stateful resource may be implemented as a logical projection over data constructed or composed dynamically from programming language objects (such as a J2EE EJB Entity Bean) or from data returned by executing a command on a private communications channel to a traditional procedural application or data system.

7.2.2. Rationale

Stateful resources can be managed via Grid service regulation. However, service regulation introduces other problems such as low efficiency and performance bottleneck. On the other hand, the real-time transaction processing is a key and challenging technology to prevent systems from various failures in Grid service framework.

Once a system possesses the primary mechanisms of system adaptation and complexity hiding, it will be able to exhibit a range of attributes, such as autonomy in the control and management of the resources inside of the system and the service provisions outside of the system [90].

This reengineering approach follows the analysis of previous chapter on analyses and represents legacy system resources by extracting and transforming useful legacy assets into Grid oriented XML components. Then, the extracted useful resources are deployed in the Grid environment as stateful resources to create Grid services. These stateful resources can be isolated and fashioned into components which can be integrated in Grid service oriented architectures. Different from traditional Grid services development process, this reengineering approach focuses on reusing the legacy resources and evolving legacy software systems into Grid services environment.

7.3. Grid Service Description

7.3.1. Stateful Resource Description

A resource properties document collects resource property elements, associated with a Web service's WSDL 1.1 portType definition to provide the declaration of the exposed resource properties of the WS-Resource. It represents a particular composed structural view or projection of the resource properties of the WS-Resource, essentially exposing the stateful resource component within the WS-Resource composition. This may be used by a service requestor to form an XML-based query or update expression on the WS-Resource. Resource property elements are almost identical to service data elements. The only difference is that resource property element declarations are simply XML global element declarations.

The WS-Resource Properties specification defines the type and values of those components of a WS-Resource's state that can be viewed and modified by service requestors through a Web service interface. In WS-Resource Properties, there is a set of more specific operations for getting and setting resource properties: single-element get, multi-element get/set, and XPath query. This specification does not dictate the means by which a service implements a resource properties document.

A given service implementation may choose to realise its implementation of the resource properties document as an actual XML instance document, stored in memory, in the file system, in a database or in some XML Repository. Other service implementations may dynamically construct the resource property elements and their values from data held in programming language objects (such as a J2EE EJB Entity Bean) or by executing a command on a private communications channel to a physical resource.

Stateful resource is defined by a single XML Global Element Declaration (GED) in a given namespace, comprising a set of references to XML GEDs of the individual resource properties. A specific resource's state may be implemented as an actual XML document and participated as services resource.

To evolve legacy systems into Grid services, the stateful resource properties document must be defined using the following rules:

1. The resource properties document must be a global element declaration (GED) in some XML namespace. This GED defines the type of the root element of a resource properties document and the type of the resource properties documents as well.

2. The complex type defining the resource properties document must only define children elements. The child elements must be aggregated using xsd:sequence or xsd:all. The order of appearance of the resource properties within the resource properties document does not matter to WS-Resource Properties.

3. The complex Type defining the resource properties document must define a sequence of one or more child elements, called resource property elements. Child elements must be defined using XML schema element reference (@ref).

4. The complex Type defining the resource properties document allows open element content (xsd:any).

Figure 7.1 shows the stateful resource properties of the De Montfort University MLE system which has been analysed in Chapter 6.

```
<wsdl:definitions ?   xmlns:tns= "http://example.com/MLE" ...>
...
  <wsdl:types>
   <xsd:schema targetNamespace="http://example.com/MLE" ... >

   <!-- Resource property element declarations -->
   <xsd:element name="Sort" type="xsd:string"/>
   <xsd:element name="Description" type="xsd:string" />
   <xsd:element name="Code" type="xsd:string" />
   <xsd:element name="Title" type="xsd:string" />
   <xsd:element name="Aim" type="xsd:string" />
   <xsd:element name="Information" type="xsd:string" />

   <!-- Resource properties document declaration -->
   <xsd:element name="MLEProperties">
     <xsd:complexType>
       <xsd:sequence>
         <xsd:element ref="tns:Sort"/>
         <xsd:any minOccurs="0" maxOccurs="unbounded" />
         <xsd:element ref="tns:Description" />
         <xsd:any minOccurs="0" maxOccurs="unbounded" />
         <xsd:element ref="tns:Code" />
         <xsd:element ref="tns:Title"
         <xsd:element ref="tns:Aim" />
         <xsd:any minOccurs="0" maxOccurs="unbounded" />
         <xsd:element ref="tns:Information" />
         <xsd:any minOccurs="0" maxOccurs="unbounded" />
       </xsd:sequence>
     </xsd:complexType>
   </xsd:element>
...
   </xsd:schema>
  </wsdl:types>
...
    <!-- Association of resource properties document to a portType -->
    <wsdl:portType name="MLE"
      wsrf-rp:ResourceProperties="tns:MLE" >

      <operation name="start" .../>
      <operation name="stop" .../>
...
...
    </wsdl:portType>
...
</wsdl:definitions>
```

Figure 7.1. Stateful Resource Properties of MLE.

In this example, the MLE system has been considered as a stateful resource. The state of "MLE" comprises six resource property components, named Sort, Code, Title, Aim, Description and Information, as well as its resource properties document, named "MLE". This simple example defines the MLE portType and the resource properties document associated with the MLE. This association between the portType and resource properties document effectively defines the type of the WS-Resource.

This document serves to define the structure upon which service-requestor-initiated query and update messages can be directed. Any operation that manipulates a resource property via the WS-Resource properties document must be reflected in the actual implementation of the WS-Resource's state.

7.3.2. Accessing Stateful Resource

The message exchange of stateful resource is based on SOAP, typically conveyed using HTTP with an XML serialisation. The state of a WS-Resource, such as the values of resource properties, can be read, modified, and queried by using standard Web service messages. These message exchanges are defined in the WS-Resource Properties specification and should be included as WSDL operations in any portType that uses the WSRP. Resource Properties is attributed to declare a WS-Resource properties document.

WS-Addressing provides transport-neutral mechanisms to address Web services and messages. This specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages. A WS-Addressing endpoint reference is an XML serialisation of a network-wide pointer to a Web service.

This pointer may be returned as a result of a Web service message request to a factory to create a new stateful resource. The patterns have been used by WS-Addressing to indicate the relationship between Web services and stateful resources. The composition of the Web services and the stateful resource can be referred as a WS-Resource.

The base functionality is to retrieve the value of a single resource property using a simple Web service request/response message exchange, identifying the WS-Resource by using a WS-Resource qualified endpoint reference as described previously and identifying the resource property by qualified name of its GED.

The GetResourceProperty request and response message must follow the implied resource pattern. The components of the GetResourceProperty request message are described as follows:

/wsrp:GetResourceProperty/QName

The contents of the GetResourceProperty response message are further as follows:

/wsrp:GetResourcePropertyResponse/{any}

The GetResourceProperty request and response message must correspond to the QName of a resource property element defined as a child of the root of the WS-Resource's resource properties document. The resource property value, such as an XML element, corresponds to the QName in the GetResourceProperty request. Figure 7.2 and Figure 7.3 show the request message and the response message of the example. They represent a request message used to retrieve two resource property elements from the WS-Resource that implements the MLE portType and the result of response value of the request. As it shows, this identifier information is carried in the SOAP header element.

```
<soap:Envelope>
   <soap:Header>
   <tns:resourceID > M L E </tns:resourceID >
   </soap:Header>
   <soap:Body>
      <wsrp:GetMultipleResourceProperty>
        xmlns:tns=   https://mles.dmu.ac.uk/ ... >
        <wsrp:ResourceProperty>
          tns: Sort
        </wsrp:ResourceProperty>
        <wsrp:ResourceProperty>
          tns: Code
        </wsrp:ResourceProperty>
      </wsrp:GetMultipleResourceProperty>
   </soap:Body>
</soap:Envelope>
```

Figure 7.2. Request Message Representation.

```
<soap:Envelope>
  <soap:Body>
    <wsrp:GetMultipleResourcePropertyResponse>
    <Sort> Course</Sort>
    </wsrp:GetMultipleResourcePropertyResponse>
    <wsrp:GetMultipleResourcePropertyResponse>
    <Code> COM502 </Code>
    </wsrp:GetMultipleResourcePropertyResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 7.3. Response Message Representation.

If the WS-Resource does not respond to the GetResourceProperty request message with the GetResourceProperty Response message, then it must send one of the following fault messages:

- Resource Unknown Fault: The resource identified in the message (which follows the implied resource pattern) is unknown to the Web service.
- Invalid Resource Property QName: The QName in the request message did not correspond to a resource property element of the WS-Resource referred in the request message.

All faults generated must be compliant with the WS-Base Faults specification.

7.3.3. Stateful Resource Notification

In an environment in which stateful resources may be created and destroyed, and may change their state dynamically, it becomes important to provide support for asynchronous notification of changes in the state of individual resources and/or other system components such as registries.

The Notification-based, interaction pattern is a commonly used pattern for inter-object communications. Examples exist in many domains, for example in publish/subscribe systems provided by Message Oriented Middleware (MOM) vendors, or in system and

device management domains. This notification pattern is increasingly used in the Web services context.

Notification is a broad concept. Not all events relate to changes in the "state" of a service or resource. WS-notification introduces a more feature-complete, generic, topic hierarchy approach for publish and subscribe based notification, which is a common model followed in large-scale distributed event management systems.

WS-resource properties then define a mapping from element names of resource properties to topic names to support functionality via its notification port-Types, which allow a client to define a subscription against one or more service data values.

From the perspective of stateful resources notification, the WS-Resource framework thus provides useful building blocks for representing and structuring notifications. From the perspective of the WS-Resource framework, the WS-Notification family of specifications extends the utility of WS-Resources by allowing requestors to ask to be asynchronously notified of changes to resource property values.

7.4. Services Implementation

The nice thing about WSDL is that it is language neutral. In other words, there is no mention of the language in which the service is going to be implemented, or of the language in which the client is going to be implemented. However, there will of course be a moment to refer to this interface from a specific language. This is carried out in the proposed approach through a set of stub classes which are generated from the WSDL file. The stub classes are usually placed in Java package. As the stubs classes are generated from the WSDL file, so they will exist after service has been compiled.

After defining the service interface, the next step is going to implement it. For referring any related stuff to a service, it is necessary to use its qualified name (QName). QName is

a name which includes a namespace and a local name. A qualified name is represented in Java using the QName class. Since the service's qualified names will be referred frequently, it is a good practice to put them all in a separate interface.

The implementation is usually being split into at least two classes: one for the service and another one for the resource. In the example depicted in Chapter 9, the two classes are defined as QNames.java and Service.java. The first one is a convenient interface class containing the QName URI/namespace constants that the service (and client) classes implemented. The second one is the service implementation that provides the core functionality for exposing local directory information.

Whenever an operation which is part of the WSDL interface has been written, the parameters and the return values in some cases will be boxed inside stub classes, which are generated automatically from the WSDL file.

7.5. Services Repository

Within the context of this thesis, service descriptions and fact specifications require a database for the persistent storage of the XML encoded service interface description. To keep the database management simple and to achieve flexibility in the service description, the proposed approach uses a table to index the service ID and the corresponding XML service description. Each fact of a service description is stored in a table. The primary key of these tables is a service ID generated during service registration. The DTD of each fact is stored in the DTD repository within the same database.

In general, the service manager retrieves the whole XML document from the database table by using traditional SQL queries. When a service is registered, the service manager can check duplicate definitions, generate the service ID, and insert the description into the database.

7.6. Grid Service Registration

For the service registration and deployment, the next step will actually make all the loose pieces which have been written available through services container. One of the key components of this phase is a file called the deployment descriptor. It is the configuration file that tells the service container how to publish the service. The deployment descriptor is written in Web Service Deployment Descriptor (WSDD) format.

The WSDD file contains information that describes one or more services. This information includes details about the back-end components that implement the operations of a service, the non-built-in data types used as parameters and return values, the SOAP message handlers that intercept SOAP messages, and so on. It is true for all deployment descriptors, WSDD file is an XML file. The WSDD of the MLE services is shown in Figure 7.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="examples/MLEService" provider="Handler" use="literal"
style="document">
        <parameter name="className" value="MLEService"/>
        <wsdlFile>share/schema/examples/MLEService/MLE_service.wsdl</wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider DestroyProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>
</deployment>
```

Figure 7.4. The WSDD of MLE Services.

7.7. Grid Service Resource Localisation

In order to implement locating the resource home for the Grid service, the Java Naming and Directory Interface (JNDI) files are deployed. JNDI is an API for directory services. It allows clients to discover and lookup data and objects via a name and, like all Java APIs that interface with host systems, is independent of the underlying implementation.

Additionally, it specifies a service provider interface (SPI) that allows directory service implementations to be plugged into the framework. The implementations may make use of a server, a flat file, or a database, the choice is up to the vendor.

JNDI defines a hierarchy of interfaces, leaving the choice of supported conformance level the provider. The API is designed to balance the genericity with extensibility. Depending on the underlying service provider implementation, the supported "object" and "attributes" types may be arbitrarily complex. The JNDI specification allows for flexibility while recommending certain minimum conformance levels that every provider should satisfy. The pluggable architecture and genericity of JNDI, coupled with logical and well-documented APIs enable relatively easy building of hierarchical Grid information systems and implementation of custom Grid service providers. The example of the JNDI file is presented in Chapter 9.

7.8. Grid Service Deployment

In Grid environment, users are able to access predefined Grid services through a high-level user friendly Grid portal. More than that, users are not only capable of using such services but can dynamically create and deploy new services in a convenient and efficient way.

To be compatible with legacy system assets, some new modules should be added to extend the original framework:

- Service oriented component repository and data repository - deposit service oriented component and database source.

- Dictionary services - supply an index of all available components and data source for system utilisation.

- Monitoring and Discovery Service (MDS) - provides information about the available resources within the Grid and their status.

- Scheduler and broker - used to invoke, control and manage all tasks when the whole system works.

- Resource management - provides the services to actually launch a job on a particular resource, check its status, and retrieve its results when it is complete.

From user's aspect, they will be supplied visualisation Grid services which enable consistent resource access across multiple heterogeneous platforms without regard for how these services are implemented. It enables mapping of multiple resource instances onto the same physical resource and facilitates management of resources within a virtual organisation.

Furthermore, it enables composing basic services to form more sophisticated services and underpinning the ability to map common service semantic behaviour seamlessly onto native platform facilities.

Figure 7.5 shows a basic services deploying structure which efficiently exploit the stateful resources in Grid environment. The communication between stateful resources and the execution services are based on XML via SOAP and HTTP. The notification component can be viewed as an independent activity. It handles subscription requests to monitor a particular resource's state.

When a resource changes to a state that matches a subscription request, the Grid service client receives an asynchronous response via this component. The other key components in the framework include service group, lifetime management and base fault, which are explained as follows.

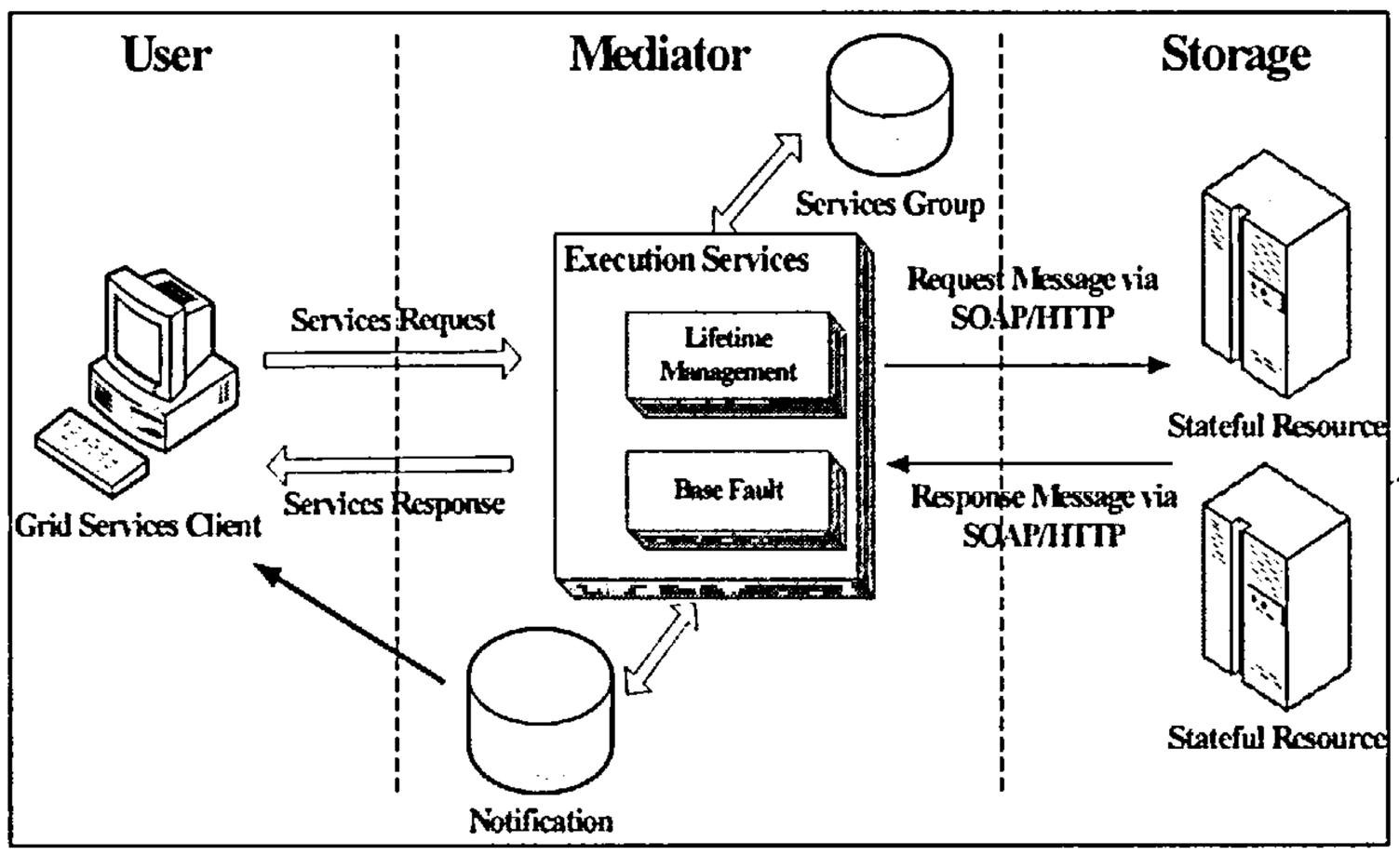

Figure 7.5. Service Deployment Structure.

7.8.1. Service Group

The term service group refers to a standard mechanism for creating a heterogeneous by-reference collection of services or resources. The services group component takes the responsibility to aggregate services with other services as a service group. It can easily perform operations such as adding new service to group, removing this service from group, and finding a service in the group.

The WS-Service Group specification [50] defines a means by which Web services and WS Resources can be aggregated or grouped together for a domain specific purpose. It expresses service group membership rules, membership constraints, and classifications using the resource property model from WS-Resource Properties. In order to form

meaningful queries for requestors against the contents of the service group, membership in the group must be constrained in some fashion. The constraints for membership are expressed by intension using a classification mechanism.

7.8.2. Lifetime Management

The lifetime component monitors each WS-Resource Property and updates the Resource Property state following a set Resource Property request. As Grid services are stateful and dynamic, the lifetime within the services is non-trivial.

The WS-Resource Lifetime specification [51] defines equivalent message exchanges. A service requestor that wishes to explicitly destroy a WS-resource must use the appropriate WS-resource qualified endpoint reference to send a destroy request message to the Web service identified by the endpoint reference. WS-Resource Lifetime defines two ways of destroying a WS-Resource: immediate and scheduled, which provides designers with the flexibility to design how their Web service applications can clean up resources which are no longer needed.

7.8.3. Base Faults

The base fault takes the responsibility to report faults when something goes wrong during a WS-Service invocation. The WS-Base Faults specification [52] defines an XML Schema type for a base fault, along with the rules for how this fault type is used by Web services.

To addresses this issue, a base XML schema definition must be defined and associated with semantics for fault messages. This definition simplifies problem determination by having a common base set of information that all fault messages contain. Note that the approach simply defines the base format for fault messages without modifying the WSDL fault message model.

7.9. Summary

This chapter presents a framework that integrates the XML packaged legacy system components into the Grid service environment. Different from the related studies, the proposed approach in this thesis describes a method for defining the legacy resources as stateful resources, and building the Grid services based on these reusable resources.

There are four major steps to migrate components in the new Grid services environment. During the first step, the services' properties and their interfaces are defined by WSDL. Then the implementation of the service is carried out by Java. In the third step, the WSDD and JNDI file define the deployment parameters including services registration and resources localisation. The last step is to deploy the Grid service. This Grid service oriented reengineering methodology brings more flexibility, expansibility and reusability, and more reliability as well.

Grid can supply a server infrastructure to achieve economies of scale, e-utilities requirement that can be easily customised on demand to meet specific customer needs and an IT infrastructure. It supports dynamic resource allocation in accordance with service level agreement policies, efficient sharing and reuse of the IT infrastructure at high utilisation levels, and distributing security from the network edge to application and data servers. And it also can deliver consistent response times and high levels of availability which in turn drive a need for end to end performance monitoring and real time reconfiguration.

This kind of service can be used as the way of solving compute-intensive problems. It can handle the computing in the different way from other types of computing which involve sharing managed resources among institutions.

Grid technology can connect services in a convenient way, but the definition of services should be more widely. The actual implementation of the service can be in hardware,

software, or both. And a service can be exported to other communities, thus providing interaction between two or more isolated communities.

Chapter 8

Extension to Semantic Grid

To meet the future of Grid, this proposed approach is extended to be used in the semantic Grid environment. Semantic Grid is an extension of the current Grid in which information and services are given well-defined meaning and better enabling computers and people to work in cooperation. It arises with the parallel development of Web services, semantic Web and Grid computing.

While both semantic Web and semantic Grid are focus on the operating in global distributed and changeable environment, there are some differences exist [89]. The semantic Web works on small number of hosts and it only provides static metadata. With regard to service oriented architecture, semantic Web supports persisting Web services and stateless Web services. On the other hand, semantic Grid works on large number of interacting processes. It provides dynamic metadata and supports transient dynamic and stateful Grid services.

Migrating legacy software systems to semantic Grid platforms will draw the significant attention in the future years, with high degree of easy-to-use and seamless automation enabling flexible collaborations and computations on a global scale. Reuse legacy systems assets in semantic Grid allows to incorporate selected parts of legacy systems to semantic Grid platforms and designs. With properties, such as information hiding, inheritance and polymorphism inherent in Grid oriented designs, essential parts of such a reengineered system can be integrated with the new semantic framework. This chapter bases on the above idea and describes a component based, reengineering approach to migrate legacy systems resources into semantic Grid framework.

8.1. Retargeting for Migration to Semantic Grid Platform

Knowledge and semantic Web technologies are evolving the Grid towards the semantic Grid to facilitate knowledge reuse and collaboration within a community of practice. Retargeting to semantic Grid builds up new systems by integration of components in the reusable library. It involves activities at the turning point between reverse engineering and forward engineering. Its retargeting involves functional restructuring and the start of forward engineering.

Normally, the users' new requirements are added on top of the existing system when the system is reengineered, and these new requirements are implemented in a small number of program functions. These functions can be ideally implemented using reusable components from the reuse library. If not, new components will have to be developed. Since adding new requirements to the new system is carried out at the specification level, this stage is called functional restructuring. In the case of the reengineering approach, this thesis is mainly concerned with integrating existing reusable components and newly developed components, and retargeting components integration in the semantic Grid platform.

8.2. Describing Grid Resource Metadata

8.2.1. Metadata in Semantic Grid

The success of semantic Grid relies on the effective discovery and seamless aggregation of required resources on the Grid. To discover and use the "right" resources for the "right" problem is not a trivial job. Users need to use not only resources' functionality information but also their own selection policies with regards to reliability, invocation

cost, provenance, quality of service etc, to determine the resources which they prefer to utilise. All such information should be precisely and consistently derived from resources' original descriptions. Consequently this requires resource providers to augment their resource descriptions with additional information. Using metadata well-informed decisions can be made on data and processes based on logical inferences. It is usually intended for consumption and interpretation by machines, rather than by humans.

Whatever the type of Grid one has, metadata or data about data is important. The term is always loosely defined and is often equivalent to information presented in "small high value records". Examples of generally important Grid metadata include user information and specifications of the resources on the Grid. These could include the CPU, memory, and number of nodes of each computer, while for software there would be the location, compiler options and perhaps specification of needed input data.

8.2.2. Grid RDF Model

Managing and operating a Grid intelligently requires the interpretation of knowledge about the state and properties of Grid components, and their configurations for solving problems. The Resources Description Framework (RDF) is a language for representing information about resources. It focuses on describing metadata on the World Wide Web.

For legacy systems, heterogeneous resources have been forced to follow the RDF to produce a very general model and syntax, which can be examined without referring to the legacy system at all. Rather than only displaying to people, RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning.

Four components have been defined in a Grid RDF Data Model. (a) Resources: a resource is anything which can be named with a Universal Resource Identifier (URI). (b) Literal: Atomic values, such as integers or strings. (c) Properties: property is a specific

aspect, characteristic, attribute, or relation used to describe a resource. (d) Statements: statement is an ordered composition of predicate, subject and object. Predicate is a property, subject is a resource, and object is a literal or a resource.

To present properties, resources and statements, W3C has defined a concrete syntax based on the XML. A specific XML mark language RDF/XML has been defined by RDF for representing RDF information and exchanging it between machines.

As a case study, the 2005 IRI conference website reused in a new semantic Grid framework. By applying this reengineering approach on the Web system, all source codes are translated and represented by XML, and restructuring is also finished during the program transformer. Then, the component based development technique is employed. The concerned resources are extracted and packaged as components which are ready to use in the future extension.

8.2.3. Implementation

This approach is applied on the IRI2005 website for example, in the IRI2005 call for papers Web page, some information can be retrieved, such as: the conference name, date, venue and sponsors, as well as the program chairs' name, address, email, phone and fax. Other unconcerned information is discarded during the reverse engineering process.

Because URI is self-extensible and it identifies all Grid resources, all Grid objects identifiers are included into a URI of the form "gird://#.#.#.#.". Then the information in the 2005 call for papers Web page (Figure 8.1) can be represented by RDF/XML. Figure 8.2 shows the RDF/XML representation of the conference information. Figure 8.3 shows the RDF/XML representation of one of the program chairs information.

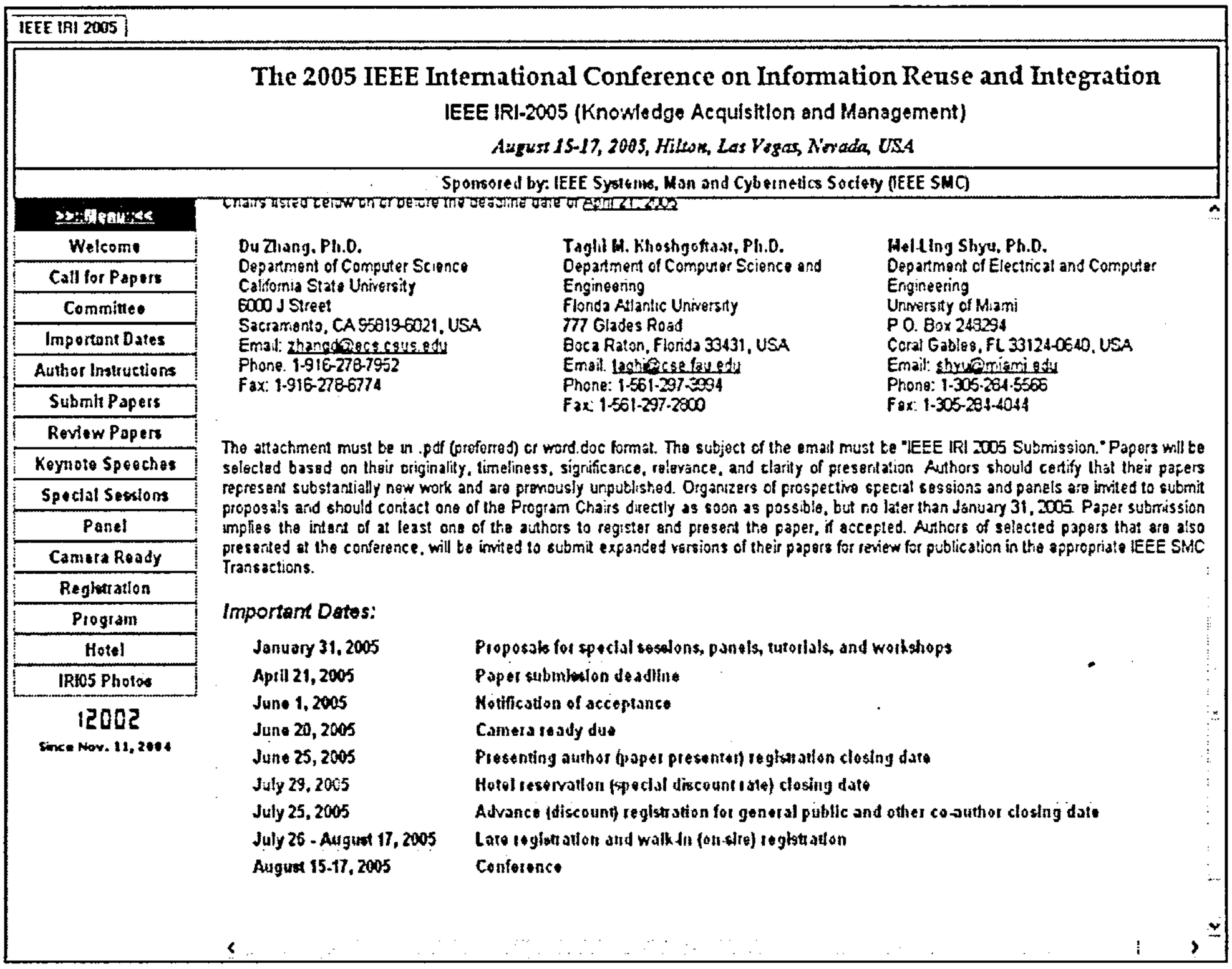

Figure 8.1. A Page of 2005IRI Conference Website.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 <rdf:Description about="grid://www.cs.fiu.edu/IRI05/>
   <g:conference>
    <g:Description about="grid://schemas/conderence">
     <g:name>The 2005 IEEE International Conference on Information Reuse and Integration</u:name>
     <g:date>Aug 15-17, 2005</u:date>
     <g:venue>Las Vegas-Nevada-USA</u:venue>
     <g:sponsors>IEEE Systems, Man and Cybernetics Society</u:sponsors>
    </g:Description>
   </g:conference>
 </rdf:Description>
</rdf:RDF>
```

Figure 8.2. RDF/XML Representation of The Conference Information.

<?xml version="1.0"?> is the XML declaration, which indicates that the following content is XML, and the version of XML.

<rdf:RDF> element indicates that the following XML content is intended to represent RDF. An XML namespace declaration is defined on the same line, presented as an xmlns attributed of the rdf:RDF start-tag. This declaration specifies that all tags in this content prefixed with rdf: are parts of the namespace identified by the URIref http://www.cs.fiu.edu/IRI05/. Beginning with the string http://www.cs.fiu.edu/IRI05/, URIrefs are used for terms from the RDF vocabulary.

The 'Description' element relates to a resources, whose URI is provided in the 'about' attribute. Then the RDF/XML representation are provided for the specific statement shown is Figure 8.1. In the conferences representation, the literal values are: 'The 2005 IEEE International Conference on Information Reuse and Integration', 'Aug 15-17, 2005', 'Las Vegas-Nevada-USA' and 'IEEE Systems, Man and Cybernetics Society'.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 <rdf:Description about="grid://www.cs.fiu.edu/IRI05/>
   <g:program chair>
    <g:Description about="grid://schemas/program chair">
     <g:name>Du Zhang</u:name>
     <g:email>zhangd@ecs.csus.edu</u:email>
     <g:phone>1-916-278-7952</u:phone>
     <g:fax>1-916-278-6774</u:fax>
    </g:Description>
   </g:program chair>
 </rdf:Description>
</rdf:RDF>
```

Figure 8.3. RDF/XML Representation of The Program Chair Information.

8.3. Migration to Semantic Grid Framework

Intent on reusing the reengineered Grid resources, a mechanism for declaring the properties and defining the relationships between these properties and other resources must be provided. RDF itself provides no means for defining such application specific classes and properties. Alternatively, such classes and properties are described as an RDF vocabulary, by using extensions to RDF provided by the RDF Vocabulary Description Language 1.0, referred as RDF Schema.

Because RDF-Schema is an instance of RDF, the syntax is also defined by the RDF XML syntax. All semantic Grid elements can be represented directly in RDF. In RDF schema, core classes are defined using rdfs: resources, rdf: property and rdfs: class. Properties are described using the RDF class rdf: property, and the RDF Schema properties rdfs: domain, rdfs: range, and rdfs: subPropertyof. So Grid Resources of 2005 IRI conference which is mentioned above as an example can be described. Figure 8.4 shows the class description and the Figure 8.5 shows the property description.

8.3.1. Class Description of Semantic Grid Resources

A basic step in any kind of description process is identifying the various kinds of things to be described. RDF Schema refers to these "kinds of things" as classes. A class in RDF schema corresponds to the generic concept of a type or category, somewhat like the notion of a class in object-oriented programming languages such as Java. RDF classes can be used to represent almost any category, such as Web pages, people, document types, databases or abstract concepts.

In RDF Schema, a class is any resource having an rdf:type property whose value is the resource rdfs:Class. So the legacy resources class would be described by assigning the

class a URIref and describing that resource with an rdf:type property whose value is the resource rdfs:Class. That is, example.org would write the RDF statement:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 <rdf:Description about="grid://classes/" ID="2005 IRI conference">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:SubClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Description>
 <rdf:Description about="grid://classes/" ID="conference information">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:SubClassOf rdf:resource="#2005IRI conference"/>
 </rdf:Description>
 <rdf:Description about="grid://classes/" ID="program chair information">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:SubClassOf rdf:resource="#2005IRI conference"/>
 </rdf:Description>
</rdf:RDF>
```

Figure 8.4. Class Description of Semantic Grid Resources.

8.3.2. Property Description of Semantic Grid Resources

In addition to describing the specific classes of things they want to describe, user communities also need to be able to describe specific properties that characterise those classes. A property is a specific aspect, characteristic, attribute, or relation used to describe a resource. The set of properties is a subset of the set of resources, that is, properties themselves are resources. RDF schema defines properties in more detail.

In RDF schema, properties are described using the RDF class rdf:Property, and the RDF schema properties rdfs:domain, rdfs:range, and rdfs:subPropertyOf. All properties in RDF are described as instances of class rdf:Property. So a new property is described by assigning the property a URIref, and describing that resource with an rdf:type property whose value is the resource rdf:Property. RDF schema also provides vocabulary for

describing how properties and classes are intended to be used together in RDF data. The most important information of this kind is supplied by using the RDF Schema properties rdfs:range and rdfs:domain to further describe application specific properties.

```
<rdf:Property rdf:ID="conference information>
 <rdfs:domain resource="grid://classes/#2005 IRI conference">
 <rdfs:range rdf:resource="#information"/>
</rdf:Property>

<rdf:Property rdf:ID="program chair information>
 <rdfs:domain resource="grid://classes/#2005 IRI conference">
 <rdfs:range rdf:resource="#People"/>
</rdf:Property>
```

Figure 8.5. Property Description of Semantic Grid Resources.

In this example, the domains of both resources are defined as 2005 IRI conference. The range of "conference information" resource is defined as "information" and the range of the "program chair information" resource is defined as "people".

8.4. Summary

Semantic Grid is by no means mature, so there is no common approach for building semantic Grid up to now. However, as it is evolved from Grid computing and the semantic Web, many researches can be commenced based on these two techniques.

This thesis proposed an approach to reusing the legacy system assets into the semantic Grid framework. The approach is based on source code translation and reconstruction, component reusing and the semantic Grid framework retargeting. The reverse engineering techniques play an important role in this analysis process.

In the proposed approach, the Grid RDF data models which are based on a specific XML mark language RDF/XML are employed to provide a simple and elegant frame for describing the properties of the reusable legacy system resources. The detailed

component mining approach needs to be tailored according to the features of a particular legacy system. RDF/XML representation and RDF schema description are key techniques in reusing recovered legacy components in semantic Grid framework.

Chapter 9

Case Study

9.1. Introduction

This chapter presents the case studies related to decomposing legacy systems, representing as Grid components and integrating Grid components into the Grid service framework. There are four core techniques proposed in this thesis for Grid service oriented legacy software systems evolution: program slicing, hierarchical cluster, XML transformation and Grid services infrastructure and framework. In this chapter, case studies have been conducted for these proposed techniques respectively. The case studies have been experimented with the proposed approach and resulting prototype.

9.2. Experimentation Software Toolkit

9.2.1. Eclipses

Eclipse was originally developed by IBM as the successor of its VisualAge family of tools. Eclipse is now managed by the Eclipse Foundation, an independent not for profit consortium of software industry vendors. It is a free software/open source platform-independent software framework for delivering what the project calls "rich-client applications", as opposed to "thin client" browser-based applications. So far this framework has typically been used to develop Integrated Development Environments (IDE), such as the Java IDE called Java Development Toolkit (JDT) and compiler that comes as part of Eclipse (and which are also used to develop Eclipse itself).

Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. A large and vibrant ecosystem of major technology vendors, innovative start-ups, universities, research institutions and individuals extend, complement and support the Eclipse platform.

The Eclipse tool integration platform is a very popular, very extensible, well-documented IDE that can be configured to host all of the useful development activities from coding to deployment to debugging. The Eclipse IDE can be used to manage all of these artifacts within a single project abstraction and coordinate all of the useful development activities from coding to deployment to debugging [98]. This thesis will use its Java project abstraction to manage the artefacts, such as: source files, WSDLs, client and server stubs, deployment configuration files, etc.

9.2.2. Globus Toolkit 4

The Globus Toolkit is an open source software toolkit used for building Grid. It is a fundamental enabling technology for the "Grid", letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy.

The toolkit is being developed by the Globus Alliance and many others all over the world. It includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. In addition to being a central part of science and engineering projects that total nearly a half-billion dollars internationally, the Globus Toolkit is a substrate on which leading IT companies are building significant commercial Grid products. A growing number of projects and companies are using the Globus Toolkit to unlock the potential of Grid for their cause.

The Globus Toolkit, currently at version 4 (GT4), is an open source toolkit for building Grid based application. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organisation has unique modes of operation, and collaboration between multiple organisations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when.

The GT4 is a WSRF compliant set of software components from which developers can build Grid systems. Because of The Globus Alliance's depth of experience in the Grid and distributed system fields and the widespread use of previous versions of the Globus toolkit, GT4 is proposed to be the premier enabling technology for Grid services. In this thesis, GT4 together with the Eclipse IDE are employed to develop, deploy, and debug a simple stateful Grid service calculator service.

9.2.3. Tomcat and Sysdeo Tomcat Launcher

Apache Tomcat (formerly under the Apache Jakarta Project) is a Web servlet container developed at the Apache software foundation. Tomcat implements the Java servlet and the Java Server Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a Web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. In this case study, Tomcat is used as the Web service container for the GT4 WSRF Web application.

The Sysdeo Tomcat Launcher is an Eclipse plug-in necessary for managing the Tomcat Web container from the Eclipse IDE. Although there are several Tomcat plug-ins for Eclipse on the market, the (free) Sysdeo Tomcat Launcher is ostensibly the most popular and well known. The Sysdeo Tomcat Launcher adds several menu-bar buttons to Eclipse for the starting and stopping of the embedded Tomcat container, and also registers the Tomcat process with the Eclipse debugger.

9.3. Components Identification

To demonstrate the evolution approach, the proposed approach has been applied to a bank management system as shown in Appendix A. This free licence software is provided by MYCPLUS for academic purpose with the source code, which is free to use, to modify and to be changed as requirements.

This is an automated software system for Bank Management, which can handle accounts of customers. It uses files to handle the daily transactions, account management and user management.

This Bank Management System performs the following functions:

- Create Individual Accounts
- Manage existing Accounts
- View daily transactions

This bank management system is not a complete accounting software just like implemented in the banks, but it still can manage the accounts of the customers using the files at backend. In this case study, this bank management system is nominated as a legacy bank system, which intended to be reengineeringed and evoloved into the Grid services environment.

9.3.1. Program Slicing Analysis

The legacy bank system is written in C language and consists 22 functions or about 1,263 lines of code. It composed of a set of programs related through external calls and it can be represented by asset of modules and a set of call relationships between modules, where each module is represented by a circle and each call relationship is represented by a line connecting two modules. The graph nodes represent the programs and it sedges depict the call relation between programs. This representation is referred as system call graph.

Program or subsystems in this bank system could be structured in a set of subroutines related through internal calls. The C languages provide primitives to explicitly define subroutines and to express internal calls. The call relation on the subroutines of a program can be represented by a graph whose nodes correspond to the program subroutines and edges depict the internal calls. This representation is referred as a program call graph. This call graph architecture describes abstracted components in the legacy system, which encapsulate valuable functionalities and are reusable in the target application or service, as well as their relationship.

According to analysis of structure of this C language legacy bank system, the system skeleton can be presented as Figure 9.1. As the fully implemented system is quite complex, only parts of them are present here.

```
Program 1
   Call Program2
   Call Program3
   Call Program4
   Call Program5
   Call Program6
End Program 1
      Program2
         Call Program7
      End Program2
            Call Program7
            End Program7
      Program3
            Call Program8
      End Program3
```

```
        Program8
            Call Program12
        End Program8
            Program12
            End Program12
Program4
    Call Program8
End Program4
Program5
    Call Program9
    Call Program10
    Call Program11
End Program5
    Program9
    End Program9
    Program10
        Call Program13
        Call Program14
    End Program10
            Program13
                Call Program15
            End Program13
                    Program15
                    End Program15
            Program14
            End Program14
    Program11
    End Program11
Program6
    Call Program11
End Program6
```

Figure 9.1. System Skeleton of Legacy Bank System.

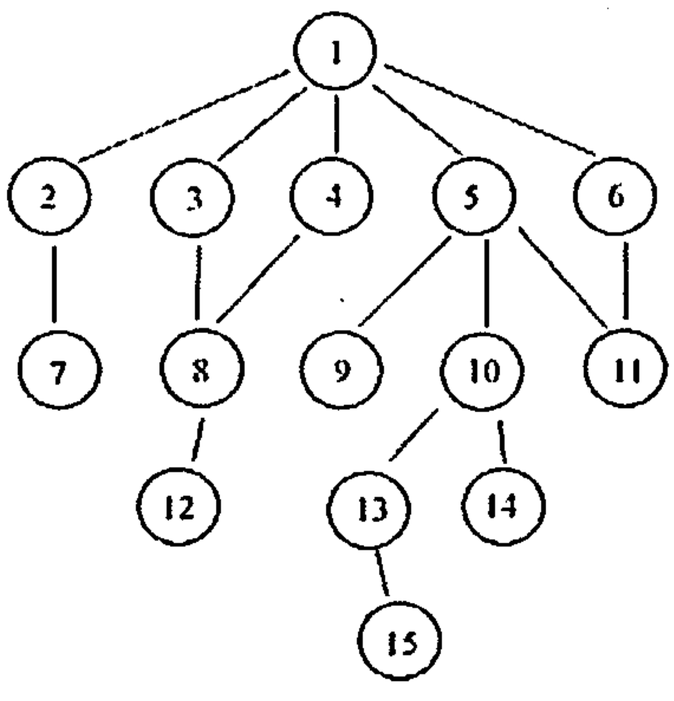

Figure 9.2. System Call Graph of Legacy Bank System.

Figure 9.2 shows the system call graph representation of the legacy bank system. The bank system includes many subsystems, such as: create account system (1, 2, 7), view

daily transactions system (1, 5, 6, 11) and manage existing accounts system (1, 3, 4, 5, 8, 9, 10, 12, 13, 14, 15).

In manage existing accounts system, there are two subsystems exist, the current account system (1, 3, 4, 8, 12) and credit card system (1, 3, 4, 5, 8, 9, 10, 12, 13, 14, 15). In this example, the credit card subsystem is the target of reusing from the legacy bank system. Module 1, 5, 9, 10, 13, 14, 15 consist of a slice under the constraint in which the start node is node1 and the end nodes are node 9, node 14 and node 15. The program call graph of the credit card subsystem is shown as Figure 9.3.

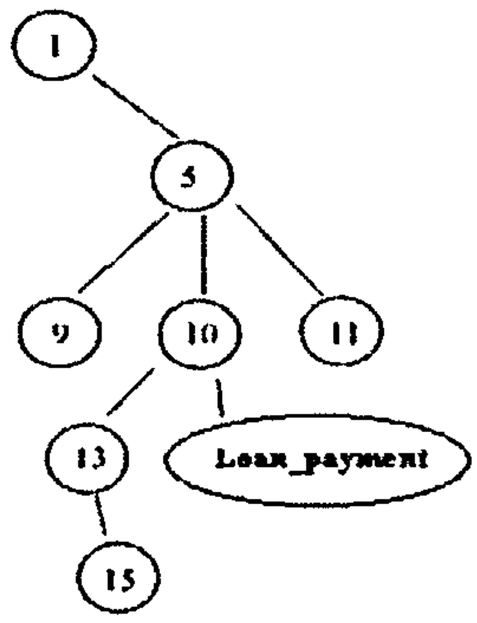

Figure 9.3. Program Call Graph of Credit Card Subsystem.

The module 14 is the loan payment subroutine in the credit card system. It is used to calculate the total repayment amount of a single credit card. The original program fragment of loan payment subroutine is shown as Figure 9.4.

```
(1)   float essential;
(2)   float interest;
(3)   float payment;
(4)   float withdraw;
(5)   interests = interest /100/12;
(6)   payment = payment *12;
(7)   withdraw=0;
(8)   x = pow (1* interests, payment);
(9)   monthly = (essential *x* interests ) / (x-1);
(10)    withdraw= withdraw+i;
(11)  if ((0<monthly) && (monthly < MAXLIMIT))
        { payments = monthly; total = monthly * payment;
          totalinterest = (minthly*payment - essential;}
(12)  else { payment = 0, total = 0, totalinterest = 0;}
```

```
(13)  printf ("%d",withdraw);
(14)  write (payment)
```

Figure 9.4. Original Program Fragment of Loan Payment Subroutine.

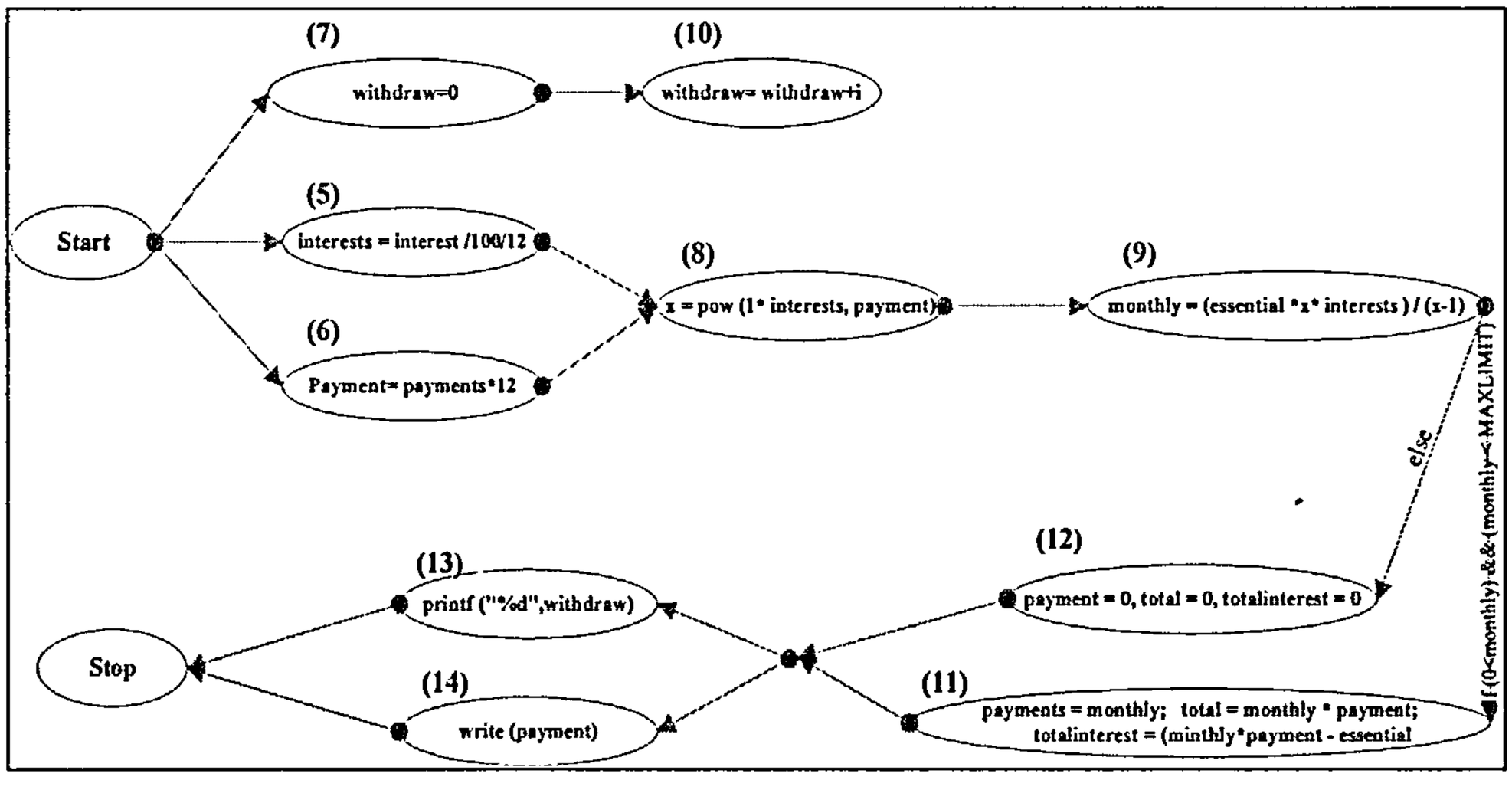

Figure 9.5. Control Flow Graph of Loan Payment Subroutine.

The Figure 9.5 presents the control flow graph of loan payment subroutine. Node 11 is flow dependent on node 9 because: (a). node 9 defines variable monthly, (b). node 11 references variable monthly, and (c). there exists a path 8->9->11 without intervening definitions of monthly. Node 11 is control dependent because there exists a path 8->9->11, and Node 12 is control dependent because there exists a path 8->9->12. Node 11 and node 12 are post-dominated by node 5, but they are not post-dominated each other.

Figure 9.6 shows the slice of the loan payment subroutine with respect to criterion (14, {payment}). It is obtained from the code in Figure 9.4 by including only those statements that could affected the value of the variable payment at line 14. As can be seen in the figure, all computations involving variable withdraw have been slided away. The deleted

statements have not effected upon the slicing criterion about payment when the program is executed in any initial states.

```
(1)   float essential;
(2)   float interest;
(3)   float payment;
(4)
(5)   interests = interest /100/12;
(6)   payment = payment *12;
(7)
(8)   x = pow (1* interests, payment);
(9)   monthly = (essential *x* interests ) / (x-1);
(10)
(11)  if ((0<monthly) && (monthly < MAXLIMIT))
        { payments = monthly;     total = monthly * payment;
          totalinterest = (minthly*payment - essential;}
(12)  else { payment = 0, total = 0, totalinterest = 0;}
(13)
(14)  write (payment)
```

Figure 9.6. Slice of Loan Payment Subroutine with Respect to Criterion (14, {payment}).

Based on the slicing approach, finally, the legacy bank system programs is deeply understand and they have been decomposed into several concerned legacy code segments. The dead codes are eliminated from the legacy system and the selected code segments function is independent.

9.3.2. Hierarchical Cluster Analysis

In order to group the sliced legacy code segments and create a hierarchical structure of them, the cluster analysis is applied. The clustering analysis is carried out to identify legacy functionalities, such as project creation, discussion forum, e-publishing and so on. Architects supervise the clustering analysis process and select the cutting point.

Figure 9.7 shows the cluster analysis of legacy bank system and Figure 9.8 shows its dendrogram output. This clustering method together with human supervision provides a powerful analysis on legacy systems, and represents them into modules.

		intro()	main_menu	new_account	new_account_gui	getaccount_no	sav_account	update_main	process	list_all	print_it	list_all_gui	trans_menu	trans_gui	show_trans	mod_trans	see_it	report	dep_gui	deposite	mod_choice	withdraw	mod_menu
		E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16	E17	E18	E19	E20	E21	E22
intro()	E1	1																					
main_menu	E2		1																				
new_account	E3			1	1	1	1		1														
new_account_gui	E4				1																		
getaccount_no	E5					1																	
sav_account	E6						1	1															
update_main	E7							1															
process	E8		1	1					1	1			1										
list_all	E9									1	1	1											
print_it	E10										1												
list_all_gui	E11											1											
trans_menu	E12								1				1	1	1		1				1		
trans_gui	E13													1									
show_trans	E14														1								
mod_trans	E15															1							
see_it	E16																1	1					
report	E17																	1					
dep_gui	E18																		1				
deposite	E19								1										1	1	1	1	1
mod_choice	E20															1				1			
withdraw	E21																					1	
mod_menu	E22																						1

Figure 9.7. Cluster Analysis of Legacy Bank System.

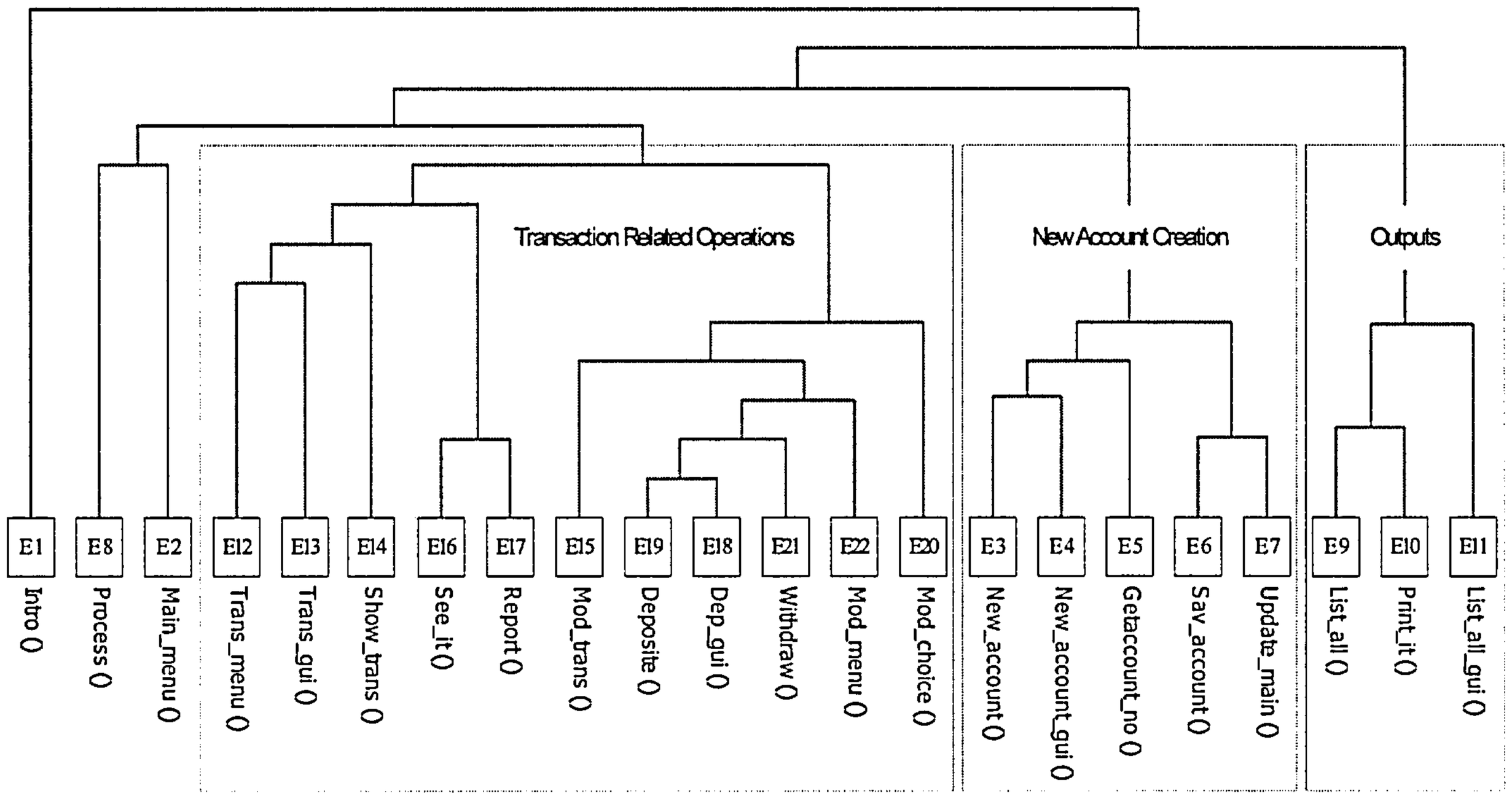

Figure 9.8. Dendrogram Output of Legacy Bank System.

9.4. Grid Component Migration

In the proposed approach, a structure of the AST has been defined to develop Grid components. By recursively traversing the hierarchy of the Grid component entities, it is able to map the Grid component to a DTD. Specifically, the tree hierarchical structure of the Grid component is mapped into XML elements and attributes. Each node and edge in the AST is mapped to an XML element tag. The attribute values of an AST node are mapped to the corresponding attribute values of the XML elements.

9.4.1. Legacy Asset Representation

In AST, each node is represented by a structure of the particular type of node. A node is created by invoking a function which returns a pointer to a structure representing that node. The function takes as arguments of subnodes for the particular type of node. For example a interests node can be created using function node "interests". The syntax for payment is:

interests = operationName [attributes] arguments responses

In this bank system, the syntax for payment is shown as:

interests = interest/100/12

Therefore, the interests node will have subnodes including the operation name, attributes, arguments and responses. The abstract syntax tree representation for "interests = interest / 100 / 12" is shown as Figure 9.9.

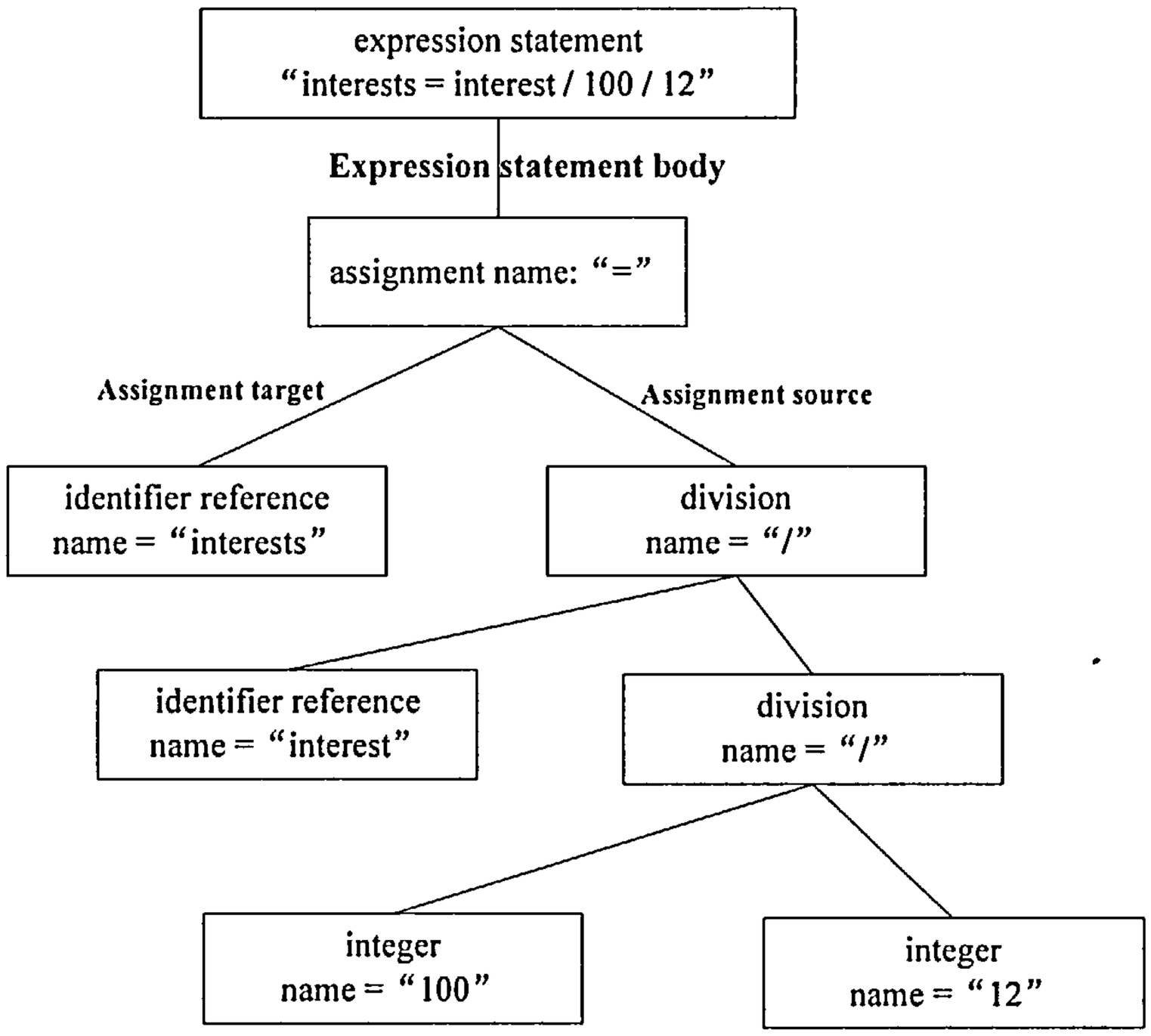

Figure 9.9. Abstract Syntax Tree Representation for "interests = interest / 100 / 12".

In this example, the non-terminal nodes are expression statement, assignment, and multiplication. The leaf nodes represent the terminal tokens, such as identifier reference integer. The edges represent the attributes as mappings between AST nodes.

For example, the edge, named assignment source, is considered as an attribute of the assignment node that contains a value node of type division. So, the loan payment subroutine can be represented by AST as Figure 9.10.

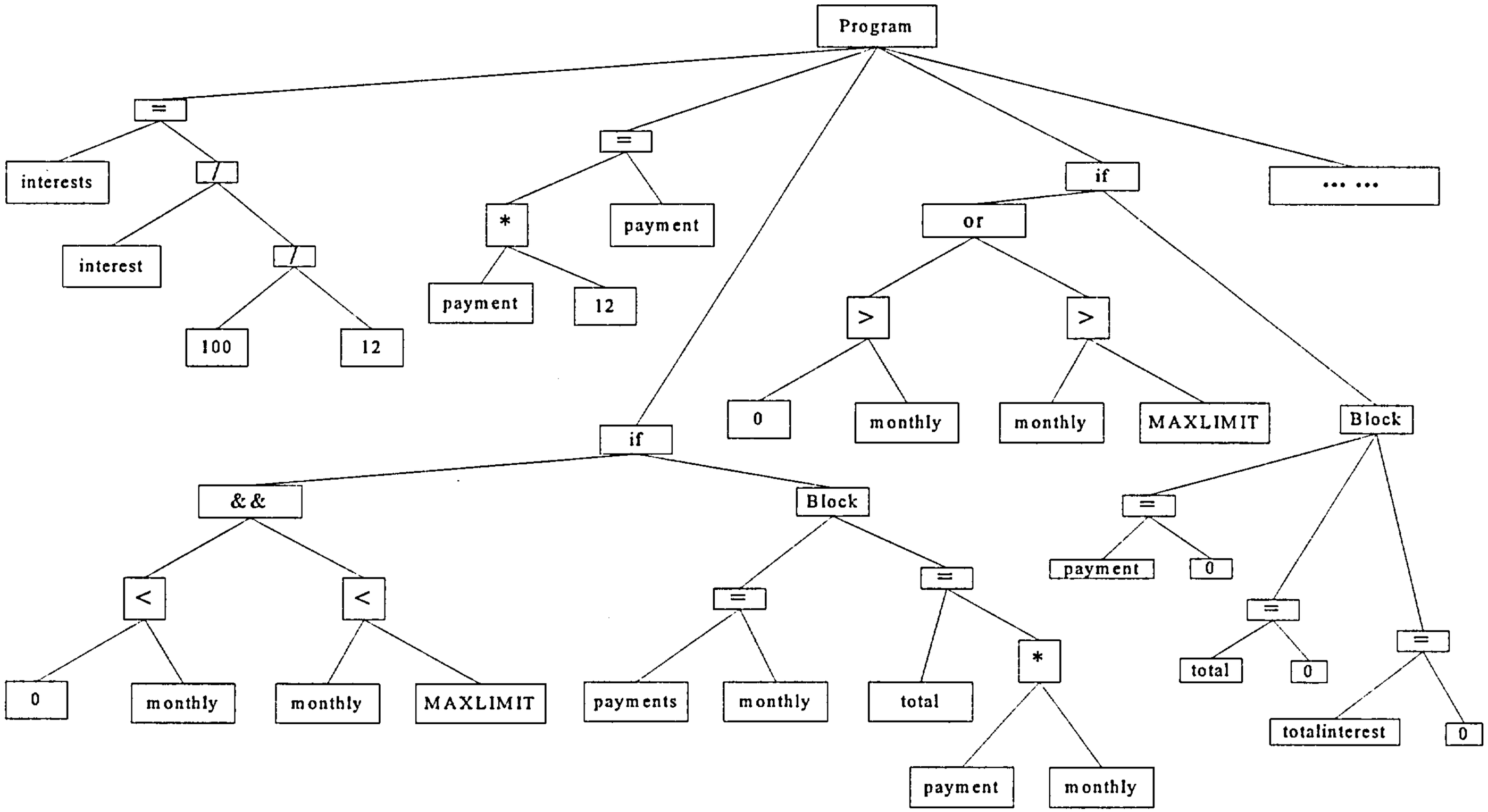

Figure 9.10. Abstract Syntax Tree Representation for Loan Payment Subroutine.

In document type definition, elements are the main building blocks of XML documents. Attributes provide extra information about elements. Entities are variables used to define common text. PCDATA means parsed character data. Character data can be seen as the text found between the start tag and the end tag of an XML element. PCDATA is text that will be parsed by a parser.

Tags inside the text are treated as markup and entities will be expanded. Otherwise, CDATA is text that will not be parsed by a parser. Tags inside the text are not be treated as markup and entities will not be expanded. The DTD representation of C expression statement are summarised in [153] as Figure 9.11 shown. So the loan payment subroutine can be represented by DTD as Figure 9.12 shown.

```
<!ELEMENT EXPRESSION-STATEMENT (EXPRESSION-STATEMENT-BODY)>
<!ELEMENT EXPRESSION-STATEMENT-BODY (EXPRESSION)>
<!ELEMENT                                                    EXPRESSION
          (EXPRESSION*|INT-LITERAL*|FLOAT-LITERAL*|STRING-LITERAL*|CHAR-LIT
          ERAL*|ASSIGNMENT-EXP*|IDENTIFIER-REF*|DATA-TYPE?)>
<!ELEMENT ASSIGNMENT-EXP (ASSIGNMENT*|SUBSTRACTION-ASSIGNMENT*)>
<!ELEMENT ASSIGNMENT (ASSIGNMENT-SOURCE, ASSIGNMENT-TARGET)>
<!ELEMENT SUBSTRACTION-ASSIGNMENT(ASSIGNMENT-SOURCE, ASSIGNMENT-TARGET)>
<!ELEMENT ASSIGNMENT-SOURCE (EXPRESSION)>
<!ELEMENT ASSIGNMENT-TARGET (EXPRESSION)>
<!ATTLIST EXPRESSION machine-dependent CDATA #REQUIRED
                     type-error CDATA #REQUIRED>
<!ATTLIST IDENTIFIER-Reference id-name CDATA #REQURIED
                               refer-to CDATA #REQURIED
                               identifier-ref-usage CDATA #REQUIRED
                               identifier-ref-via CDATA #REQUIRED>
<!ATTLIST INT-LITERAL int-long CDATA #REQUIRED
                      int-radix CDATA #REQURIED
                      int-unsigned CDATA #REQURIED
                      int-value CDATA #REQURIED>
<!ATTLIST STRING-LITERAL    string-value CDATA #REQUIRED>
<!ATTLIST CHAR-LITERAL      char-value CDATA #REQURIED>
```

Figure 9.11. DTD Representation of C Expression Statement.

```
<!ELEMENT Legacy Bank System (Credit Card System)>
<!ELEMENT Credit Card System (Loan Payment)>
<!ELEMENT Loan Payment (Interests,Payment)>
<!ELEMENT Interests (Interest,Division)>
 <!ATTLIST interests = interest/100/12 CDATA #ReQuired>
 <!ATTLIST interests, interest CDATA #ReQuired>
```

```
  <!ATTLIST 100, 12 CDATA #ReQuired>
<!ELEMENT Payment (Payment,Multiplication)>
  <!ATTLIST payment = payment *12 CDATA #ReQuired>
  <!ATTLIST payment CDATA #ReQuired>
  <!ATTLIST 12 CDATA #ReQuired>
```

Figure 9.12. DTD Representation of Loan Payment Subroutine.

9.4.2. XML Component

For the purpose of importing existing XML represented information sources to Grid services domain, the interfaces are essentially defined in a complex XML type. It is attributes to the name, type and name space. The XML grammar can be used to identify all publicly-available methods, their signatures, and their return types.

In the proposed approach XML is used to describe the structure of the data, XSL language is used to manipulating XML from one structure into another. And Java is used to encapsulate them with a few simple classes and implement the application. For legacy systems which are not written in Java, wrap technique is applied to supply a Java interface for them.

As the adopted legacy bank system is C language, to encapsulate reusable legacy C code by Java, a common interface is implemented via Java Native Interface. Java Native Interface (JNI) [61] is an API that allows Java code to interact with code written in another language. Using JNI to integrate native methods with Java may not always be an optimal or elegant solution, but it is necessary when slices legacy modules are not immediately available in Java.

Wrapping legacy code can also result in better performance than pure Java code, though this benefit will presumably diminish as a variety of compiler and runtime techniques continue to close the performance gap between legacy code and Java code [72]. Another advantage of this approach is that it is less risky and highly transport as it requires no

change to legacy code in the upper layers, and it is very powerful. Also it could reduce expenses of the entire project.

The proposed method creates a JNI wrapper for reusable legacy code includes three steps:

- The first step is to define a new class that extend and overrides the original method. The java system routers all drawing operations for a new object through the original method, as it does for all other GUI objects.

- The next step is to generate a header file that describes the interface in the native method that Java expects to be used. Necessary corresponding between Java and other language files is created automatically to make native calls.

- The final step is to write the native rendering methods with an interface that conforms to the header file that generated, and built it as a standard shared library. Then the user can initiate and control all these actions through a simple GUI.

```
JNIEXPORT jfloatArray JNICALL
Java_LoanImp_Loan_Payments (JNIEnv *env, jobject obj, jfloatArray arr)

{
    jsize len = ( *env)->GetArrayLength(env.arr);
    jfloatArray sum;
    jfloat *jarry =    ( *env)->GetFloatArrayElements(env,arr,0);

/*float Loan_payments (float essential, float interests, float payment )(
original function call - (no longer necessary)*/
   essential = body [0];
   interests =body[1]; /*Data stream IN*/
   payment = body [2];

    interests = interest /100/12;
    payment = payment *12;
    x = pow (1* interests, payment);
    monthly = (essential *x* interests ) / (x-1);
    if ((0<monyhly) && (monthly < MAXLIMIT)) {
        payments = monthly;
        total = monthly * payment;
        totalinterest = (minthly * payment) - essential; }
    else { payment = 0, total = 0, totalinterest = 0; }

    body [0] = payments;
    body [1] = total;
    body [2] =totalinterests;
```

```
    jarry [0] = payments;
    jarry [1] = total;
    jarry [2] =totalinterests;
    sum = (*env)-> NewfloatArray (env, len*sizeof(float)+1);
    (*env)->SetFloatArrayRegion (env, sum, 0, 10 * sizeof (float),
    (jfloat *) jarray);
    (*env)-> ReleaseFloatArrayElements(env, arr, jarray, 0);
    return sum; /*Data Stream OUT*/
}
```

Figure 9.13. Wrapping Loan Payment Subroutine with JNI.

Figure 9.13 shows the example of wrapping loan payment subroutine in the legacy bank system with JNI technique. The reusable legacy code in C language (Loan_payments function) is in charge of calculating the total repayment amount in a credit agreement when it is supplied with a number of parameters. The reference to the Loan_payments function is defined as an interface. It is treated as a class to specify an instance in the legacy code, and then the functionality of the Loan_payments function is embedded in the target application. The essential JNI glue code is illustrated.

```
<application>
    <network>
     <instance componentName="Source"
         componentPackage="Loan_Payments" id="1">
              <property name="degrees of freedom" value="100"/> </instance>
     <instance componentName="Payment"
          componentPackage="Payment" id="2"/>
     <dataflow sinkComponent="2" sinkPort=" interests"
          sourceComponent="1" sourcePort="interests"/>
        ...
    </network>
</application>
```

Figure 9.14. XML Component Represented Application.

Figure 9.14 shows its XML Component represented application. The application builder produces a XML Component application description document. It consists of the <network> information. The network XML data represents a composition of component instance "source" and "payment" together with any users.

```
<repository>
<component package="bank.Loan_payments" name="Source">
<propertyDefinition type=" " name=" " value=" "/>
<port objectPackage="bank.payments" objectName="payment" portName="payment"/>
<implementation language="C" platform="C" url="file:.">
<action portName="payments">
<binding method="getpayments"> ... </binding>
<classPerformanceModel type="initial" url="http:" />
</action>
</implementation>
<implementation language="C" platform="C" url="file:."> ... </implementation>
</component>

<object package="bank.Loan_payments" name="payment" >
<method name="getpayments" type="action">
<argument typeName="payments" typePackage="bank.Loan_payments" />
</method>
</object>
</repository>
```

Figure 9.15. XML Component Represented Repository.

Figure 9.15 shows its XML Component represented repository. The repository XML Component data provides the interface information for the <component> types, specifying <port> and <property> elements. The types and default properties specified in the repository XML Component allow customisation where is required.

The repository component information also indicates component inheritance and package information. The repository also contains information needed to create the run time representation, namely the meta-data for the methods in the component implementation. These are represented by <implementation> and <action> elements. The <action> elements specify the bindings of ports and the location of corresponding performance data. The package and location information for the executables are stored alongside the implementation data in the repository. These are referred to <object> elements.

9.5. Grid Service Integration

This section discusses developing and deploying a simple Globus toolkit 4 Grid service. The GT4 service uses WSRF to keep stateful information, and it is developed by eclipse

and tomcat. The eclipse is used to help the developer handle all tedious files such as source files, WSDL files and XML files, and it is also helpful in the configuration deploy steps.

To build Grid services is a tedious work, it may include switch between many tools such as editors, command shells, file managers, build tools, application containers, etc. in the development process. With the plug-ins and configuration, the Eclipse IDE can be used to manage all of these tasks within a single project abstraction and coordinate all of the useful development activities from coding to deployment to debugging. By embedding the Apache Tomcat services container within Eclipse, any update to the Grid service implementation can be immediately reflected in the active running Grid service.

Previous sections show the process of component identification and Grid component migration. To perform a cooperate Grid service in this section, the components identification, and migration approach has been applied on a calculator system. The system is decomposed and interested components are migrated. As a result, the concerned addition and subtraction Grid components are retrieved.

The proposed Grid service is named as Calculator Service (CalculatorService). It integrates the loan payment components from the legacy bank system, with the addition, subtraction function components from a legacy calculator system. It may allowsusers to perform the following operations:

- Calculate the total payment of all credit cards per month.
- Calculate the paying and payout monthly.
- Assist to manage personal financial affairs.

Furthermore, Calculator Service will have the following resource properties (RP):

- Loan Payment (string)

- Interests (string)
- Payment (string)
- Value (integer)
- Last operation performed (string)
- GetValue (this operation is used to access the Value RP)

In this Grid service, once a new resource is created, the "value" RP is initialised to zero, and the "last operation" RP is initialised to "none". The addition and subtraction operations expect only one integer parameter. This parameter is added/subtracted to the "value" RP, and the "last operation" RP is changed to "addition" or "subtraction" accordingly.

This service also can be extended by add more functions. This prototype is just a means to prove the proposed approach in this thesis. Typical Grid services are generally much more complex and could perform more operations than it does in this case study.

9.5.1. Service Resource Description

The first step in writing a Grid service is to descript the service resources. This step do not concerned with the inner workings of that service, it focuses on specifying what the service provides and indicating what operations will be available to the clients. The service interface is called port type (portType). The WSRF in this part is used to keep the state of resources.

The service resources description is performed by WSDL, although it maybe a bit harder to understand than some other interface definition languages (such as Java interface). The main reason to choose WSDL is that, although Java interfaces are easier to write and understand, in the long run they may produce much more problems than WSDL does. Considering that, the sooner the WSDL is used, the better the service works.

The Calculator.wsdl file is the XML document that describes the calculator service interface. It should be placed in the schema/examples/CalculatorService project folder. The service interface describes how the outside world can interact with this service, specifically the operations that can be performed on it. Figure 9.16 shows the WSDL code of the CalculatorService.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CalculatorService"
targetNamespace=http://examples.strl.org/calculator/CalculatorService

xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://examples.strl.org/calculator/CalculatorService"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wslw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
      namespace=
      "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
      location="../../wsrf/properties/WS-ResourceProperties.wsdl" />

<!== Types == >
<types>
<xsd:schema targetNamespace="http://examples.strl.org/calculator/CalculatorService"
xmlns:tns="http://examples.strl.org/calculator/CalculatorService"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <xsd:import
            namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
            schemaLocation="../../ws/addressing/WS-Addressing.xsd" />

<!== Requests and Responses == >
      <xsd:element name="add" type="xsd:int"/>
      <xsd:element name="addResponse">
            <xsd:complexType/>
      </xsd:element>

      <xsd:element name="subtract" type="xsd:int"/>
      <xsd:element name="subtractResponse">
            <xsd:complexType/>
      </xsd:element>

      <xsd:element name="getValueRP">
            <xsd:complexType/>
      </xsd:element>
      <xsd:element name="getValueRPResponse" type="xsd:int"/>

<!== Resources Properties == >

      <xsd:element name="LegacyBankSystem" type="xsd:string"/>
      <xsd:element name="CreditCardSystem" type="xsd:string"/>
      <xsd:element name="LoanPayment" type="xsd:string"/>
```

```
        <xsd:element name="Interests" type="xsd:string"/>
        <xsd:element name="Payment" type="xsd:string"/>
        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>
.....
        <xsd:element name="CalculatorResourceProperties">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
                    <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>

<!== Messages == >
<message name="AddInputMessage">
<part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
<part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
<part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
<part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="GetValueRPInputMessage">
<part name="parameters" element="tns:getValueRP"/>
</message>
<message name="GetValueRPOutputMessage">
<part name="parameters" element="tns:getValueRPResponse"/>
</message>

<!== Porttype == >
<portType name="CalculatorPortType"
wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="tns:CalculatorResourceProperties">

<operation name="add">
<input message="tns:AddInputMessage"/>
<output message="tns:AddOutputMessage"/>
</operation>

<operation name="subtract">
<input message="tns:SubtractInputMessage"/>
<output message="tns:SubtractOutputMessage"/>
</operation>

<operation name="getValueRP">
<input message="tns:GetValueRPInputMessage"/>
<output message="tns:GetValueRPOutputMessage"/>
</operation>

</portType>
</definitions>
```

Figure 9.16. WSDL Document of CalculatorService.

The <portType> element defines three operations: add, subtract, and getValueRP, along with all the necessary messages and types. All operations consist of an input and an output message, described with the above <message> tags.

The wsrp:ResourceProperties attribute of the portType element are used to specify what the service's resource properties are. The resource properties must be declared in the <types> section of the WSDL file. The resource properties are used to keep all state information.

With the help of the wsdlpp:extends attribute of the portType element, existing WSRF portTypes can be included in the own portType without having to copy-and-paste from the official WSRF WSDL files. A WSDL Preprocessor will use the value of that attribute to generate correct WSDL which includes the own portType definitions plus any WSRF portType which might need in the service.

9.5.2. Implement Service

9.5.2.1. The QNames Interface

The implementing services step answers the question of how the service performs the operations that proposed. The service implementing is done with Java. Qualified name (QName) is a name which includes a namespace and a local name which is used to refer to just about related entity to a service. For example, the QName of the Value RP is:

{http://examples.strl.org/calculator/calculator service}Value

This is a common string representation of a QName. The namespace is placed between curly braces, and the local name is placed right after the namespace.

A qualified name is represented in Java using the QName class. Since the service's qualified names will be referred frequently, it is a good practice to put them all in a separate interface:

As Figure 9.17 shows, the CalculatorQNames.java file is a convenient interface class containing the QName URI/namespace constants relevant to the proposed Grid service. It should be placed in the org/strl/examples/services/Calculator/impl project folder. By having the service classes implement this interface, these constants can be referred to replicate themselves throughout the project.

```
package org.strl.examples.services.Calculator.impl;
import javax.xml.namespace.QName;
public interface CalculatorQNames
{
        public static final String NS = "http://examples.strl.org/calculator/calculator service";
        public static final QName RP_VALUE = new QName(NS, "Value");
        public static final QName RP_LASTOP = new QName(NS, "LastOp");
        public static final QName RESOURCE_PROPERTIES = new QName(NS,"calculatorResourceProperties");
}
```

Figure 9.17. QNames.java Document of CalculatorService.

The GetResourceProperty request and response message correspond to the QName of a resource property element. The components of the GetResourceProperty request message are further described as follows: */wsrp:GetResourceProperty/QName*

The contents of the GetResourceProperty response message are further described as follows: */wsrp:GetResourcePropertyResponse/{any}*

Figure 9.18 and Figure 9.19 show the request message and the response message of the example. They represent a request message used to retrieve two resource property elements from the WS-Resource that implements the loan payment portType and the result of response value of the request. As it shows, this identifier information is carried in the SOAP header element.

```
<soap:Envelope>
      <soap:Header>
      <tns:resourceID> Loan Payment </tns:resourceID>
      </soap:Header>
      <soap:Body>
            <wsrp:GetMultipleResourceProperty>
                xmlns:tns=     https://... >
                <wsrp:ResourceProperty>
```

```
                tns: Interests
            </wsrp:ResourceProperty>
            <wsrp:ResourceProperty>
                tns: Payment
            </wsrp:ResourceProperty>
        </wsrp:GetMultipleResourceProperty>
    </soap:Body>
</soap:Envelope>
```

Figure 9.18. Request Message Representation.

```
<soap:Envelope>
    <soap:Body>
        <wsrp:GetMultipleResourcePropertyResponse>
        <Interests> "interests"</Interests>
        </wsrp:GetMultipleResourcePropertyResponse>
        <wsrp:GetMultipleResourcePropertyResponse>
        <Payment> "payment"</Payment>
        </wsrp:GetMultipleResourcePropertyResponse>
    </soap:Body>
</soap:Envelope>
```

Figure 9.19. Response Message Representation.

9.5.2.2. The Service Implementation

In this case study, the service implementation consists of a single Java class with the code for both the service and the resource. It can also be split into two classes: one for the service and another one for the resource.

The CalculatorService.java file is the service implementation that provides the core functionality for exposing local directory information. It should be placed in the org/strl/examples/services/Calculator/impl project folder. The source for this Java class is shown as Figure 9.20.

```
package org.strl.examples.services.Calculator.impl;
import java.rmi.RemoteException;

import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
```

```
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;

import org.strl.examples.Calculator.CalculatorService.AddResponse;
import org.strl.examples.Calculator.CalculatorService.SubtractResponse;
import org.strl.examples.Calculator.CalculatorService.GetValueRP;

public class CalculatorService implements Resource, ResourceProperties
{
	/* Resource Property set */
	private ResourcePropertySet propSet;

	/* Resource properties */
	private int value;
	private String lastOp;

	/* Constructor. Initialises RPs */
	public CalculatorService() throws RemoteException {
		/* Create RP set */
		this.propSet = new SimpleResourcePropertySet(CalculatorQNames.RESOURCE_PROPERTIES);

		/* Initialise the RPs */
		try {
			ResourceProperty valueRP = new ReflectionResourceProperty(CalculatorQNames.RP_VALUE,
		"Value", this);
			this.propSet.add(valueRP);
			setValue(0);
			ResourceProperty lastOpRP = new ReflectionResourceProperty(
			CalculatorQNames.RP_LASTOP, "LastOp", this);
			this.propSet.add(lastOpRP);
			setLastOp("NONE");
		} catch (Exception e) {
				throw new RuntimeException(e.getMessage());
		}
	}

	/* Get Setters for the RPs */
	public int getValue() {
		return value;
	}
	public void setValue(int value) {
		this.value = value;
	}
	public String getLastOp() {
		return lastOp;
	}
	public void setLastOp(String lastOp) {
		this.lastOp = lastOp;
	}

	/* Remotely-accessible operations */
	public AddResponse add(int a) throws RemoteException {
		value += a;
		lastOp = "ADDITION";
		return new AddResponse();
	}

	public SubtractResponse subtract(int a) throws RemoteException {
		value -= a;
		lastOp = "SUBTRACTION";
		return new SubtractResponse();
	}
	public int getValueRP(GetValueRP params) throws RemoteException {
```

```
        return value;
        }

        /* Required by interface ResourceProperties */
        public ResourcePropertySet getResourcePropertySet() {
                return this.propSet;
        }
}
```

Figure 9.20. Service Implementation Document of CalculatorService.

9.5.3. Define Deployment Parameters

9.5.3.1. The JNDI Deployment File

To enable the implementation to locate the resource home for the Grid service, the Java Naming and Directory Interface (JNDI) files are deployed. JNDI is an API for directory services. It allows clients to discover and lookup data and objects via a name and, like all Java APIs that interface with host systems, is independent of the underlying implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

<service name="examples/core/first/CalculatorService">
        <resource name="home" type="org.globus.wsrf.impl.ServiceResourceHome">
        <resourceParams>

                <parameter>
                        <name>factory</name>
                        <value>org.globus.wsrf.jndi.BeanFactory</value>
                </parameter>

        </resourceParams>

        </resource>
</service>
</jndiConfig>
```

Figure 9.21. JNDI Deployment Document of CalculatorService.

The deploy-jndi-config.xml file is the JNDI deployed file that enables the GT4 WSRF implementation to locate the resource home for this service. It should be placed in the /org/strl/examples/services/CalculatorService project folder. The source for this configuration file is shown as Figure 9.21.

9.5.3.2. The WSDD Deployment Descriptor

To enable Grid service for client connections, the next step is to make all the loose pieces which have been created available through a Web services container. Deployment descriptor is the configuration file that tells the services container how to publish the service. The deployment descriptor is written in WSDD format (Web Service Deployment Descriptor). The deploy-server.wsdd file of the CalculatorService is shown in Figure 9.22. It should be placed in the /org/strl/examples/services/ CalculatorService project folder.

The service name <*service name="examples/CalculatorService" provider="Handler" use="literal" style="document"*> specifies the location of the service. If it is combined with the base address of services container, the full URI of the service could be obtained. For example, if the GT4 standalone container is used, the base URL will probably be *http://localhost:8080/wsrf/services.* Therefore, the service's URI would be: *http://localhost:8080/wsrf/services/examples/CalculatorService.*

The className <*parameter name="className" value="org.globus.examples.services. Calculator.impl.CalculatorService"/*> refers to the class which implements the service interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
      xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <service name="examples/CalculatorService" provider="Handler" use="literal" style="document">
            <parameter name="className"
      value="org.globus.examples.services.Calculator.impl.CalculatorService"/>
```

```
            <wsdlFile>share/schema/examples/CalculatorService/CalculatorService.wsdl</wsdlFile>
            <parameter name="allowedMethods" value="*"/>
            <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
            <parameter name="scope" value="Application"/>
            <parameter name="providers" value="GetRPProvider"/>
            <parameter name="loadOnStartup" value="true"/>
        </service>
</deployment>
```

Figure 9.22. WSDD Deployment Document of CalculatorService.

In the WSDL file *<wsdlFile>share/schema/examples/CalculatorService/Calculator Service.wsdl</wsdlFile>*, the wsdlFile tag tells the services container where the WSDL file for this service can be found. This WSDL file (CalculatorService.wsdl) will be generated automatically by a GT4 tool when the service is compiled.

9.5.4. Service Deployment

9.5.4.1. Create Ant Launch Configuration

This section presents process of creating the Ant launch configuration necessary for building and deploying the Grid service. A launch configuration is a mechanism provided by Eclipse for executing one or more Ant buildfile targets.

Ant, an Apache Software Foundation project, is a Java build tool. It allows programmers to forget about the individual steps involved in obtaining an execution from the source files, which will be taken care of by Ant. Each project is different, so the individual steps are described in a buildfile. This buildfile directs Ant on what it should compile, how it should compile, and in what order.

Grid Archive(GAR) file is a single file which contains all the files and information the service container needs to deploy service and make the service available to the internet. Creating a GAR file include (a) processing the WSDL file to add missing pieces (such as bindings), (b) creating the stub classes from the WSDL, (c) compiling the stubs classes,

(d) compiling the service implementation and (e) organising all the files into a very specific directory structure. This could be done by Ant.

The buildservice.xml Ant buildfile serves as the "master" buildfile that coordinates activities from the globus-build-service and WS-Core buildfiles. The globus-build-service distribution is an Ant buildfile (and related shell scripts) to assist with the building of GAR files.

It is a general-purpose Ant buildfile and script that give a set of Java, WSDL, WSDD files conforming to a specific directory structure, and will generate a GAR file without the need to manually edit the Ant file. This buildfile is a variation of the one used in the GT4 Programmer's Tutorial [130]. The buildservice.xml Ant buildfile file should be placed in the project root directory. This buildfile is shown as Figure 9.23.

This "master" buildfile calls the following external buildfiles:

- The Globus-Build-Service build.xml Ant buildfile (/dev/GTK/etc/)
- The WS-Core build-packages.xml Ant buildfile (/Dev/GTK/share/globus_wsrf_common/build-packages.xml)
- The WS-Core tomcat.xml Ant buildfile (/Dev/GTK/share/globus_wsrf_common/tomcat/tomcat.xml)

```
<?xml version="1.0"?>
<project default="all" name="Grid Service Buildfile" basedir=".">
    <description>
        Grid Service Buildfile
    </description>
    <property environment="env"/>
    <target name="all">
        <ant antfile="${build.gar}" target="clean"/>
        <ant antfile="${build.gar}" target="all"/>
        <ant antfile="${build.packages}" target="deployGar"/>
        <ant antfile="${build.tomcat}" target="deploySecureTomcat"
    dir="/Dev/GTK/share/globus_wsrf_common/tomcat" />
    </target>
</project>
```

Figure 9.23. Buildservice.xml Document of CalculatorService.

Before creating a launch configuration for this buildservice.xml buildfile, a buildservice.properties file need to be created. The buildservice.properties file contains the "name=value" properties specific to this service and needed to guide the various build tasks through the GAR creation and deployment process.

This file should be added to the project's root directory. The document for this file is shown as Figure 9.24.

```
package=com.strl.examples.services.calculator
interface.name=Calculator
package.dir=org/strl/examples/services/Calculator
schema.path=examples/CalculatorService
service.name=CalculatorService
gar.filename=org_strl_examples_calculator

build.gar=/dev/GTK/etc/build.xml
build.packages=/Dev/GTK/share/globus_wsrf_common/build-packages.xml
build.tomcat=/Dev/GTK/share/globus_wsrf_common/tomcat/tomcat.xml
gar.name=/Dev/eclipse/workspace/ProvisionDirService/org_strl_examples_calculator
tomcat.dir=/Dev/tomcat5
```

Figure 9.24. Buildservice.properties Document of CalculatorService.

Relying on the globus-build-service script and buildfile allows creating a GAR file with minimal effort, and without having to modify an Ant buildfile every time when move on to the next example.

With this Ant buildfile which is now linked to the project and a corresponding buildservice.properties file which, a launch configuration that will create the .GAR for this service now can be created, as well as some other necessary files.

9.5.4.2. Building and Deploying

Up to now, all necessary files have been added, the package explorer in eclipse navigator view should looks like Figure 9.25.

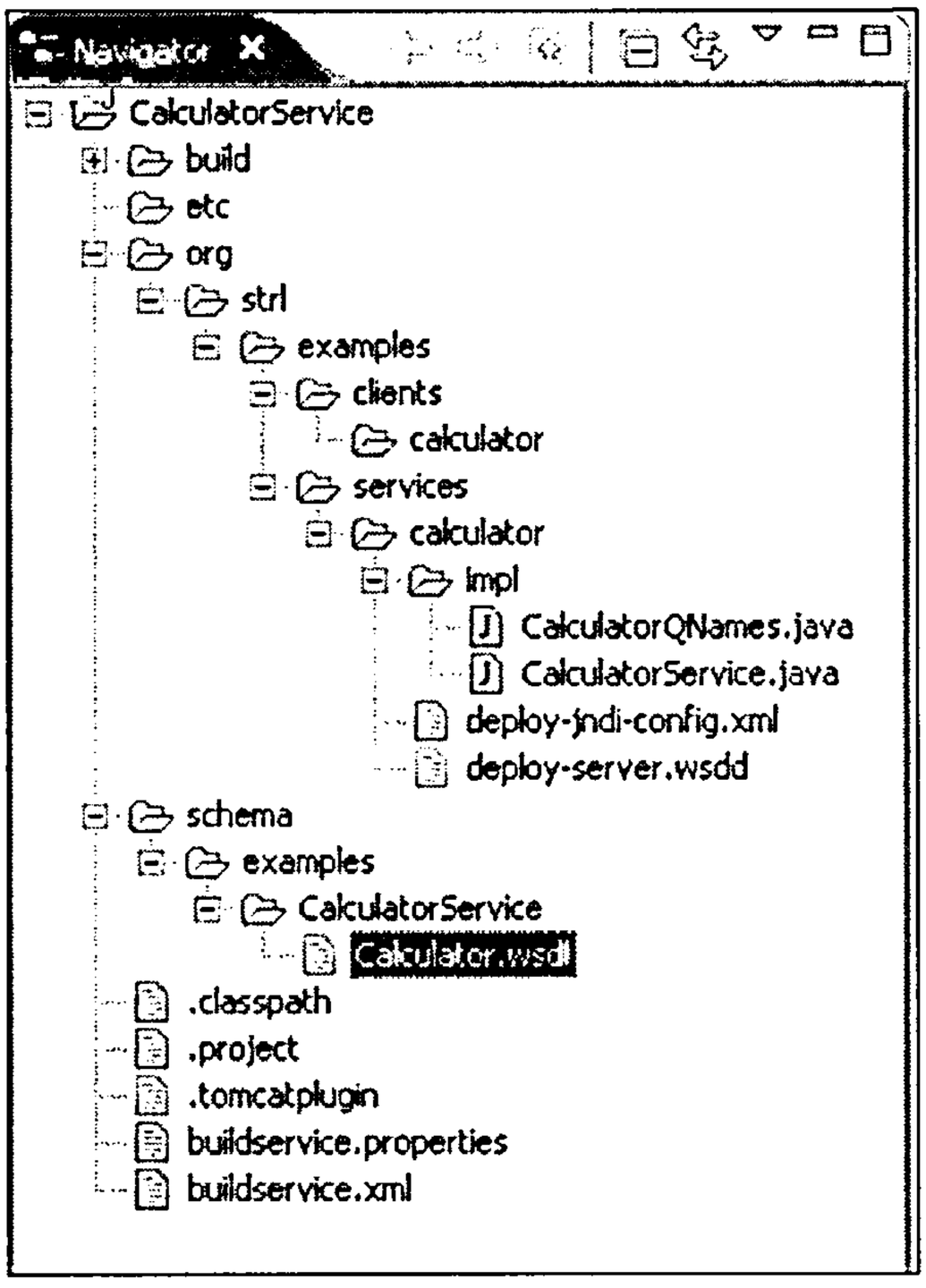

Figure 9.25. Updated Package Explorer.

Using the provided Ant buildfile and the handy script, building a service can be done by doing the following commands:

/globus-build-service.sh -d <service base directory> -s <service's WSDL file>

In the eclipse IDE, this step can be performed by clicking on the buildservice.xml file and select "Run as" option. In the run as> external tools setup page, the buildfile is set as: *${workspace_loc:/CalculatorService/buildservice.xml*, and the base directory is set as: *${workspace_loc:/CalculatorService}* in the main option. And the property file in the properties option is set as: *${workspace_loc:/CalculatorService/buildservice.properties}*. Then the service building can be finished by executing the buildservice.xml file. This will fulfil the Ant task, generate all of the remaining necessary files, create the service GAR,

deploy the GAR into WSRF, and deploy WSRF into Tomcat. Figure 9.26 shows the build successful message in the console view.

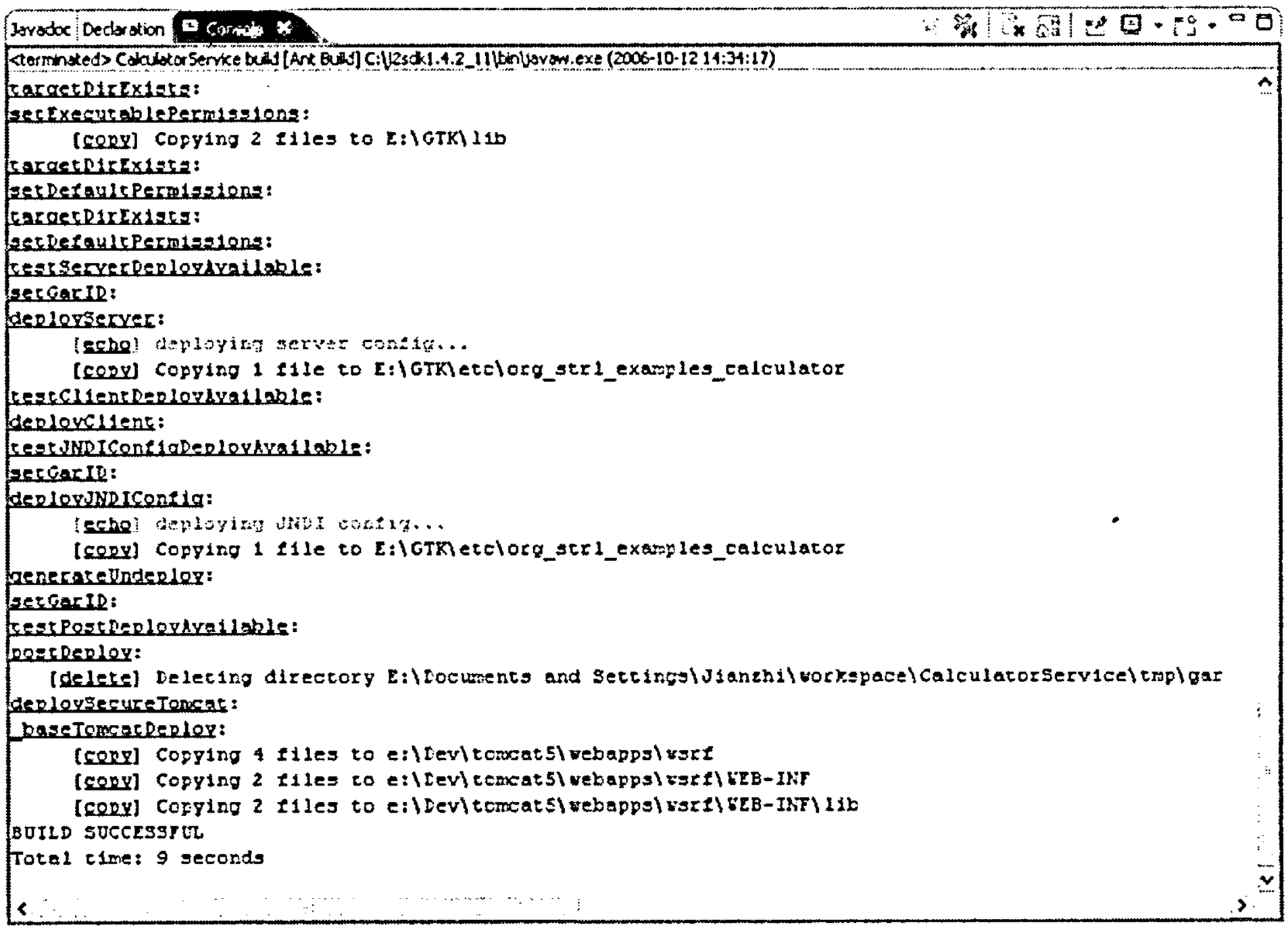

Figure 9.26. Build Successful Message.

9.5.5. Running Grid Service

The running of the proposed Grid service can be performed as starting the Tomcat container by clicking the Start/Stop/Restart Tomcat toolbar buttons.

The Apache Tomcat runs the WSRF Grid services and exposes the service to the exterior client. The output of Tomcat can monitor in the console view. Figure 9.27 shows the services running output and Figure 9.28 shows the services terminated output.

```
Javadoc | Declaration | Console
Tomcat 5.x [Java Application] C:\j2sdk1.4.2_11\bin\javaw.exe (2006-10-12 14:38:08)
: Create Host deployer for direct deployment ( non-jmx )
6-10-12 14:38:11 org.apache.catalina.core.StandardHostDeployer install
: Processing Context configuration file URL file:E:\Program Files\Apache Software Foundation\Tomcat
6-10-12 14:38:11 org.apache.catalina.core.StandardHostDeployer install
: Processing Context configuration file URL file:E:\Program Files\Apache Software Foundation\Tomcat
6-10-12 14:38:11 org.apache.struts.util.PropertyMessageResources <init>
: Initializing, config='org.apache.struts.util.LocalStrings', returnNull=true
6-10-12 14:38:11 org.apache.struts.util.PropertyMessageResources <init>
: Initializing, config='org.apache.struts.action.ActionResources', returnNull=true
6-10-12 14:38:12 org.apache.struts.util.PropertyMessageResources <init>
: Initializing, config='org.apache.webapp.admin.ApplicationResources', returnNull=true
6-10-12 14:38:14 org.apache.catalina.core.StandardHostDeployer install
: Processing Context configuration file URL file:E:\Program Files\Apache Software Foundation\Tomcat
6-10-12 14:38:14 org.apache.catalina.core.StandardHostDeployer install
: Installing web application at context path /tomcat-docs from URL file:E:\Program Files\Apache Sof
6-10-12 14:38:14 org.apache.catalina.core.StandardHostDeployer install
: Installing web application at context path /jsp-examples from URL file:E:\Program Files\Apache So
6-10-12 14:38:14 org.apache.catalina.core.StandardHostDeployer install
: Installing web application at context path /servlets-examples from URL file:E:\Program Files\Apac
6-10-12 14:38:14 org.apache.catalina.core.StandardHostDeployer install
: Installing web application at context path /webdav from URL file:E:\Program Files\Apache Software
6-10-12 14:38:14 org.apache.coyote.http11.Http11Protocol start
: Starting Coyote HTTP/1.1 on http-8080
6-10-12 14:38:15 org.apache.jk.common.ChannelSocket init
: JK2: ajp13 listening on /0.0.0.0:8009
6-10-12 14:38:15 org.apache.jk.server.JkMain start
: Jk running ID=0 time=0/62  config=E:\Program Files\Apache Software Foundation\Tomcat 5.0\conf\jk2
6-10-12 14:38:15 org.apache.catalina.startup.Catalina start
: Server startup in 4828 ms
```

Figure 9.27. Services Running Output.

```
Javadoc | Declaration | Console
<terminated> Tomcat 5.x [Java Application] C:\j2sdk1.4.2_11\bin\javaw.exe (2006-10-12 14:38:08)
6-10-12 14:40:22 org.apache.catalina.core.StandardHostDeployer remove
: Removing web application at context path /admin
6-10-12 14:40:23 org.apache.catalina.logger.LoggerBase stop
: unregistering logger Catalina:type=Logger,path=/admin,host=localhost
6-10-12 14:40:23 org.apache.catalina.core.StandardHostDeployer remove
: Removing web application at context path /webdav
6-10-12 14:40:23 org.apache.catalina.core.StandardHostDeployer remove
: Removing web application at context path /servlets-examples
6-10-12 14:40:23 org.apache.catalina.core.StandardHostDeployer remove
: Removing web application at context path /jsp-examples
6-10-12 14:40:23 org.apache.catalina.core.StandardHostDeployer remove
: Removing web application at context path /balancer
6-10-12 14:40:23 org.apache.catalina.core.StandardHostDeployer remove
: Removing web application at context path /tomcat-docs
6-10-12 14:40:23 org.apache.catalina.core.StandardHostDeployer remove
: Removing web application at context path /manager
6-10-12 14:40:23 org.apache.catalina.core.StandardHostDeployer remove
: Removing web application at context path
6-10-12 14:40:23 org.apache.catalina.logger.LoggerBase stop
: unregistering logger Catalina:type=Logger,host=localhost
6-10-12 14:40:23 org.apache.catalina.logger.LoggerBase stop
: unregistering logger Catalina:type=Logger
6-10-12 14:40:23 org.apache.coyote.http11.Http11Protocol destroy
: Stopping Coyote HTTP/1.1 on http-8080
```

Figure 9.28. Services Terminated Output.

The service deployment structure is shown as Figure 7.5. To completely fulfil the Grid service deployment, the following parts should be considered:

- Service Group - takes the responsibility to aggregate services with other services as a service group to perform operations such as adding new service to group, removing this service from group, and finding a service in the group. It is performed by WS-Service Group specification.

- Lifetime Management - monitors each resource property and updates their state following a set resource property request. It is performed by WS-Resource Lifetime specification.

- Base Fault - takes the responsibility to report faults when something goes wrong during a service invocation. It is performed by WS-Base Faults specification.

9.6. Summary

The purpose of this legacy bank system case study is to demonstrate that the proposed approach has the ability to evolve legacy systems into Grid service environment. This is achieved though the following points:

- Legacy systems analysis with program slicing and software clustering.

- Grid component migration with XML transformation and representation.

- Grid services integration includes services stateful resources description, services implementation, deployment parameters definition and services deployment.

Chapter 10

Conclusions

In the context of software evolution, legacy software systems are continuously evolved in order to correct errors, provide new functionality, or port into modern platforms. Grid is an increasingly popular topic, which emerged within a project designed to link wide-area supercomputing resources to computational and collaborative science. With the adoption of the service oriented architecture, Grid services are emerged by integrating Grid computing and Web services to perform a seamless information processing system across distributed, heterogeneous, dynamic virtual organisations that the user can access from any location. Could legacy software systems be evolved into Grid environment? The answer is "Yes". The proposed approach in this thesis integrates traditional methods with those emerging technologies:

- Component based development. Component based software engineering is a process that aims to design and construct software systems using reusable software components. It emphasis on decomposition of the engineered systems into functional or logical components with well defined interfaces used for communication across the components.

- Extensible Markup Language. XML is the de facto technique for data management and information exchange. It has become ubiquitous for all new applications. For existing legacy applications, XML is the ideal glue to integrate them with new systems.

- Grid services. Grid service have emerged by combining Web services and Grid computing to perform a seamless information processing system across distributed,

heterogeneous, dynamic virtual organisations that the user can access from any location.

- Semantic Grid. Semantic Grid is an extension of the current Grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation. It arises with the parallel development of Web services, semantic Web and Grid computing.

- Software reverse engineering techniques. In this thesis, the program slicing technique is used to decompose system, understand program, eliminate dead code and make selected code segments function independently by component interface parameters determination and deep source code comprehension. The clustering analysis technique is used to group large mounts of entities in a dataset and capture reusable legacy code segments into clusters according to their relationship and similarity from legacy systems, and create a hierarchical structure of these reusable legacy code segments.

This research aims to unify the above technologies for a solution to the issues raised in Chapter 1. Each of these technologies is given a detailed discussion on how to apply them in the proposed solution to Grid oriented legacy system evolution.

10.1. Comparison and Evaluation

As Grid is an emerging technique, there is not much work for integrating Grid technique with software evolution. From the legacy systems evolution perspective, few of them bridge software evolution and Grid development together. Most of these researches focus on the reusable resources identification from legacy systems, but they can not be integrated in the Grid system directly. Most current research works of service oriented software evolution are still stay on Web services.

Even though some researches focus on the Grid oriented evolution areas, they only concentrate on parts of reengineering process and do not provide a general framework of the legacy systems evolution towards Grid environment. Moreover, some of them are even based on the unfashionable standards such as Open Grid Services Architecture and Open Grid Services Infrastructure. Generally speaking, current approaches about reengineering legacy systems with Grid technology are not mature and accomplished, novel approaches are required in this research area.

Different from the related studies, a general framework is proposed in this thesis for the Grid oriented legacy software evolution which includes the components identification used in Grid environment, Grid components packing and integration for Grid services steps. The proposed approach is also extended to the semantic Grid environment for the semantic Grid oriented legacy systems evolution.

The techniques such as program slicing and software clustering, the component based development methods and the up to date standards WSRF are used in this approach to analyses legacy systems, define legacy resources as Grid components and stateful resources and build Grid services with these reusable resources. The proposed approach in this research extends previous researches on software evolution and Grid application development, and integrates them to provide a unified framework for Grid oriented legacy software system evolution.

Affected by the Grid orientation trend, many existing software systems will turn into legacy systems. These legacy systems require Grid oriented reengineering, which can facilitate legacy system evolution in Grid service orientated architecture. Comparing design a new system, evolving legacy system into Grid environment could massively reduce time and cost for the enterprise. And it will bring more flexibility, expansibility and reusability.

10.2. Summary

Grid oriented legacy software system evolution is a challenge in the new ear of software engineering. This thesis develops an effective approach to evolve legacy software systems into Grid environment. In particular, first, reverse engineering techniques are used for program comprehension and design recovery. Then the legacy software systems are decomposed into a hierarchy of subsystems by defining relationships between the entities of the underlying paradigm of the legacy system. The decomposition is driven by program slicing and clustering techniques. Next, Grid components are created by wrapping objects and defining the interface. Finally, Grid components are allocated to Grid services environment by specifying the requirements of the system and characteristics of the network as an integer programming model. The aim of this approach is to use legacy systems into Grid environment which enables the integration of legacy resources with Grid across distributed, heterogeneous, dynamic environment and communities. Around this theme, several significant technical issues were addressed:

Legacy system decomposition and component identification. Because the complete reverse engineering of legacy systems is difficult to archive and the cost of reengineering the whole legacy system is expensive, to reuse recovered legacy components is an economic and efficient way to reengineer legacy systems. This component based Grid oriented reengineering methodology is feasible, and it contributes on reengineering legacy software systems into Grid services components. This approach is modestly invasive, but it offers a high return on investment for legacy assets and many benefits for the Grid service development.

XML representation and Grid component composition. Once a software component has been extracted from a legacy system, or has been built as a new component, its interface can be extracted and represented in XML. The XML representation is not only finished by the component wrap, but also finished with the sources code analysis included such as

the using of AST, DTD and XSLT. The term "Grid component" refers to a collection of legacy reusable resources. It describes the relation of legacy resources and the semantic rules governing the resources. A Grid component is theoretically independent of its representation schema. The resulting components with core legacy code function in Grid service framework for the intent of sharing distributed resources and coordinated problem solving.

Grid service integration. To achieve the Grid service integration, different from the related studies, the proposed approach describes a method for defining the legacy resources as stateful resources, and builds the Grid services based on these reusable resources. Four steps are summarised to migrate Grid components in the new Grid services environment. In the first step, the service's properties and its interface are defined by WSDL. Then the implementation of the service is carried out by Java. In the third step, the WSDD and JNDI file define the deployment parameters include service registration and resources localisation. The last step is to deploy the Grid service. This Grid service oriented reengineering methodology brings more flexibility, expansibility and reusability, as well as more reliability. All in all, legacy systems can be extracted and reengineered into stateful resources by Grid oriented evolution approach. These stateful resources can be isolated and integrated into Grid service architectures. The proposed approach contributed on evolving a legacy software system into Grid services by an improved reverse engineering method.

Semantic Grid oriented extension. Although semantic Grid is not mature, it is an evolution of Grid computing and the semantic Web, therefore many researches can be commenced based on these two techniques. This proposed semantic Grid oriented legacy software system evolution approach is based on source code translation and reconstruction, component reusing and the semantic Grid framework retargeting. The reverse engineering techniques play an important role in this analysis process. The semantic Grid is an extension of the current Grid in which information and services are

given well defined and explicitly represented meaning. To reuse legacy software systems in the semantic Grid environment, Grid RDF data models which are based on a specific XML mark language RDF/XML are employed to provide a simple and elegant frame for describing the properties of the reusable legacy system resources. RDF/XML representation and RDF-schema description are key techniques to reuse recovered legacy components in semantic Grid framework.

10.3. Assessment of Success

The research results meet the success criteria given in Chapter 1 as follows:

Can this approach handle the diversity of Grid based systems?

In this research, the proposed approach employ reverse engineering techniques to decompose legacy software systems and represent these concerned resources by XML. To reuse legacy software systems in the Grid services environment, the XML represented legacy resources are defined as stateful resources. Then by using WSDL to define the resources properties and interface, using WSDD and JNDI to define the resources registration and localisation, and using Java to implement the service, the legacy resources can be successfully deployed in the Grid services.

To reuse legacy software systems in the semantic Grid environment, Grid RDF data models which are based on a specific XML mark language RDF/XML are defined to provide a simple and elegant frame for describing the properties of the reusable legacy system resources. RDF/XML representation and RDF-schema description are key techniques to reuse recovered legacy components in semantic Grid framework. The proposed component based evolution approach can handle the diversity of Grid based systems to present a unified view.

Does the legacy system component identification and Grid XML component representation methods effective and feasible for the Grid oriented legacy software system evolution process?

In this research, two reverse engineering techniques are employed to help the legacy system component identification for use in Grid environment. The program slicing technique is used to decompose system, understand program, eliminate dead code and make selected code segments function independently by component interface parameters determination and deep source code comprehension. The clustering analysis technique is used to group large mounts of entities in a dataset and capture reusable legacy code segments into clusters according to their relationship and similarity from legacy systems, and create a hierarchical structure of these reusable legacy code segments.

Then, for the Grid XML component migration and packing, by recursively traversing the hierarchy of the component entities, the tree hierarchical structure of the component is mapped into XML elements and attributes. Each node and edge in the AST is mapped to an XML element tag. The attribute values of an AST node are mapped to the corresponding attribute values of the XML elements. To accomplish this mapping, a set of transformations is defined to convey the grammar of the programming language to the Document type declarations (DTD).

DTDs are generated to describe the characteristics of the data making the documents self contained and usable as a data exchange format. Combined with client-side interpreted XSLT stylesheets, it provides an approach for publishing data from relational databases on the XML Web. Therefore, the software evolution techniques play an important role to help the Grid oriented legacy system evolution and Grid development industry. The proposed approach effectively integrate the software evolution techniques with the Grid development process.

Does this approach improve the efficiency of Grid development industry?

Grid is a brand new technology, more and more research institutes and commercial organisations are focus on the Grid application development. Mostly, legacy systems can not be simply discarded as they are critical to business they support and they encapsulate a great deal of knowledge and expertise about the application. Compared with designing a brand new system, the method of using reusable legacy resources to development Grid application based on software evolution approach is less risky and highly transparent. It requires no change to legacy code in the upper layers, and it is very powerful. Also it can massively reduce time and the expenses of the entire project. This software evolution approach improves a lot of efficiency of the Grid development industry.

Is the approach feasible for realisation? For example, is it possible to build a practical tool based on the approach?

Quite a lot attention was paid to the practical part of the approach during development. The methods of legacy component identification, XML representation and Grid services integration infrastructure and framework are not only theoretically correct, but also workable for real legacy systems. The examples and case studies show that the approach is a "practical" one, an it is feasible for practice.

Is the approach capable for industrial-scaled systems?

The approach is capable for industrial-scaled systems and efficient enough for real practice in evolving legacy systems into Grid environment. It adopts an infrastructure based on latest standard Grid technologies such as WSRF in Grid services and RDF/XML in semantic Grid which are beneficial to large scale system integration to deploy legacy resources into the Grid environment.

10.4. Conclusions

Grid is a new technology, so there is currently few active research related to evolve legacy systems within Grid environment. Most researches are necessary and significant in this area to leverage and extend legacy resources in Grid environment. From economic aspects, a business is constantly re-organising, changing its boundaries and reconfiguring its activities. Grid oriented evolution enables legacy systems to adapt continuous changing in business logic and market requirements. From technical aspects, legacy applications integration towards Grid and service will become common in Grid oriented environment. As a result, there are increasing interests in migrating and reengineering legacy software systems with these new Grid technologies and software development paradigms.

This research proposes a solution for migrating legacy software systems using a Grid user interface to enable the dynamic activation and Grid resources discovery. This ability of the legacy system to use Grid will allow the enterprise to take advantage of the broad commercial support provided by Grid technology.

Concerning on the reuse of legacy software system, because the complete recovery of legacy systems is difficult to archive and the cost of reengineering the whole legacy system is expensive, reusing is an economic and efficient way to reengineer legacy systems.

Through this research experience, it is argued that the detailed component mining approach needs to be tailored according to the features of a particular legacy system. The adopted software clustering techniques and program slicing techniques are not new, but the comprehensive analysis ascertains the reusable legacy code efficiently.

Through the discussion in this thesis, it is concluded that the legacy system evolution can assist Grid application development. The proposed legacy system evolution framework is powerful for utilising reusable legacy resources into Grid environment to build Grid

applications across distributed, dynamic environment and service oriented architecture communities.

Based on the process of system decomposition, resources representation and Grid environment integration presented in this thesis, the legacy system assets can be successfully evolved into the Grid services and semantic Grid environment. Comparing to design a brand new system, based reusing legacy systems, this Grid oriented evolution approach is feasible and it massively reduce the developing time and cost for the enterprise.

10.5. Future Work

The future work of this research may include the following issues:

- Infrastructure and Framework. For the research, the implementation of the proposed framework is enough to demonstrate the Grid oriented evolution concept. However, the framework need to be developed for meet more type of Grid applications in the future.

- Legacy system analysis and resource identification. Although a legacy system decomposition and component mining approach are proposed to identification components for using in Grid environment, there is not enough experiment done due to limited time of this research and it may not suitable for all kinds of legacy systems. For evolving full-blown legacy software systems, they need to be further developed.

- The autonomic computing [138] can be introduced to establish a framework for triggering Website reengineering process by an automatic perception of changes in the business requirements and environments.

- The section of semantic Grid oriented evolution focuses on how to retrieve the useful resources from legacy systems and represent the resources based on the semantic

Grid standards. In the future, more attention will be paid on how to run the reusable resources in the semantic Grid environment, which may include the following aspects: Expose the meaning of Grid services, resources and entities by assertions in a common RDF data model. Publish and share consensually agreed ontology in Web ontology language OWL and query, filter, integrate and aggregate the metadata in RDF Data Query Language (RDQL) [118].

- IDE support is also important to successful automation of the proposed approach. An viable solution to this is developing Plug-ins for mature production level IDEs, such as NetBeans Eclipse [98], so that the development and reengineering work can be done with the full support of IDE supplied functionalities.

Reference

[1] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. Reinefeld, F. Schintke, T. Schott, E. Seidel and B. Ullmer, "The Grid Application Toolkit: Toward Generic and Easy Application Programming Interfaces for The Grid", *Proceedings of the IEEE*, Vol. 93, No.3, pp. 534-550, 2005.

[2] D. Alur, J. Crupi and D. Malks, *Core J2EE Patterns,* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[3] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer, "SETI@ Home: An Experiment in Public-Resource Computing", *Communications of the ACM*, Vol. 45, No. 11, pp. 56-61, 2002.

[4] K. Arnold, R. Scheifler, J. Waldo, A. Wollrath and B. O'Sullivan, *A Jini Specification*, Addison Wesley Longman Publishing Company, Boston, MA, USA, 1999.

[5] R. Arnold, "Software Restructuring", *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 607-617, 1989.

[6] D. Atkinson and W. Griswold, "The Design of Whole Program Analysis Tools", *International Conference on Software Engineering (ICSE)*, 1996.

[7] D. Atkinson and W. Griswold, "Implementation Techniques for Efficient Data-Flow Analysis of Large Programs", *International Conference on Software Maintenance (ICSM)*, pp. 52-61, 2001.

[8] X. Bai, H. Yu, G Wang, Y. Ji, M. Marinescu, C. Marinescu and L. Boloin, "Coordination in Intelligent Grid environments", *Proceedings of the IEEE*, Vol. 93, No. 3, pp. 613-630, 2005.

[9] M. Baker, R. Buyya and D. Laforenza, "Grids and Grid Technologies for Wide-Area

Distributed Computing", *Software - Practice and Experience*, Vol. 32, No. 15, pp. 1437-1466, 2002.

[10] M. Baker and G. Smith, "Jini Meets the Grid", *International Conference on Parallel Processing Workshops (ICPPW)*, pp. 193-198, 2001.

[11] F. Baude, D. Caromel and M. Morel, "From Distributed Objects to Hierarchical Grid Components", *Lecture Notes in Computer Science*, Vol. 2888, pp. 1226-1242, 2003.

[12] R. Baumgartner, S. Flesca and G. Gottlob, "Visual Web Information Extraction with Lixto", *27th International Conference on Very Large Data Bases (VLDB)*, pp. 119-128, 2001.

[13] K. Bennett, "Legacy Systems: Coping With Success", *IEEE Software*, Vol. 12, No. 1, pp 19-23, 1995.

[14] L. Bent, D. Atkinson and W. Griswold, "A Comparative Study of Two Whole-Program Slicers for C", *Technical Report CS2000-0643*, University of California at San Diego, USA, 2001.

[15] H. Bergsten, *Java Server Page*, O'Reilly Publishing, ISBN 0596005636, 2003.

[16] T. Biggerstaff and C. Ritcher, "Reliability Framework, Assessment, and Directions", *IEEE Software*, Vol. 4, No. 2, pp. 41-49, 1987.

[17] D. Binkley and K. Gallagher, "Program Slicing", *Advances in Computers*, Vol. 43, pp. 1-50, 1996.

[18] T. Bodhuin and M. Tortorella, "Using Grid Technologies for Web-enabling Legacy Systems", *11th Annual International Workshop on Software Technology and Engineering Practice (STEP)*, pp. 186-195, 2004.

[19] C. Boldyreff and R. Kewish, "Reverse Engineering to Achieve Maintainable WWW Sites", *8th Working Conference on Reverse Engineering (WCRE)*, pp. 249-257, 2001.

[20] G. Booch, *Software Components with Ada*, Benjamin-Cummings Publishing Company,

Redwood City, CA, 1987.

[21] D. Booth, M. Champion, C. Ferris, F. Mccabe, E. Newcomer and D. Orchard, "Web Services Architecture", *W3C Working Draft,* 14 May 2003. Available at: http://www.w3.org/TR/2003/WD-ws-arch-20030514/#id2608426.

[22] A. Bosworth, D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, C. Kaler, D. Langworthy, F. Leymann, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, E. Sindambiwe, T. Storey and S. Weerawarana, "Web Services Addressing," *W3C Working Draft,* 2004. Available at: http://www.w3.org/Submission/ws- addressing.

[23] A. Brown, *Component Based Software Engineering : Selected Papers from the Software Engineering Institute*, Los Alamitos, CA, IEEE Computer Society Press, 1996.

[24] B. Butchart, C. Chapman and W. Emmerich, "OGSA First Impression – A Case Study Re-engineering a Scientific Application with the Open Grid Services Architecture", *UK e-Science All Hands Meeting*, 2003.

[25] G. Canfora, A. Cimitile, A. De Lucia and G. Di Lucca, "Decomposing Legacy Programs: A First Step Towards Migrating to Client-Server Platforms", *Journal of Systems and Software*, Vol. 54, No. 2, pp. 99-110, 2000.

[26] L. Chen, S. Cox, F. Tao, N. Shadbolt, C. Puleston and C. Goble, "Empowering Resource Providers to Build the Semantic Grid", *IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, China, pp.271-277, 2004.

[27] L. Chen, C. Wang and F. Lau, "A Grid Middleware for Distributed Java Computing with MPI Binding and Process Migration Supports", *Journal of Computer Science and Technology*, Vol. 18, No. 4, pp. 505-514, 2003.

[28] C. Chiang, "The Use of Adapters to Support Interoperability of Components for Reusability", *Information and Software Technology*, Vol. 45, No.3, pp. 149-156, 2003.

[29] E. Chikofsky and J. Cross, "Reverse Engineering and Design Recovery: A Taxonomy", *IEEE Software*, Vol. 7, No.1, 1990.

[30] L. Childers, T. Disz, R. Olson, M. Papka, R. Stevens and T. Udeshi, "Access Grid: Immersive Group-to-Group Collaborative Visualization", *4th International Immersive Projection Technology Workshop*, 2000.

[31] S. Comella-Dorda, K. Wallnau, R. Seacord and J. Robert, "A survey of Black-Box Modernization Approaches for Information Systems", *International Conference on Software Maintenance (ICSM)*, pp. 173–183, 2000.

[32] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, "The WS-Resource Framework Version 1.0", *White Paper,* Copyright Computer Associates International, 2004.

[33] E. Deelman, G. Singh, M. Atkinson, A. Chervenak, N. Chue Hong, C. Kesselman, S. Patil, L. Pearlman and M. Su, "Grid-Based Metadata Services", *16th International Conference on Scientific and Statistical Database Management (SSDBM)*, pp.393-402, 2004.

[34] A. Deursen and T. Kuipers, "Identifying Objects Using Cluster and Concept Analysis", *21th International Conference on Software Engineering (ICSE)*, Los Angeles, USA, pp. 246-255, 1999.

[35] G. Di Lucca, A. Fasolino, F. Pace, P. Tramontana and U. DeCarlini, "Comprehending Web Applications by a Clustering Based Approach", *10th International Workshop on Program Comprehension (IWPC)*, pp. 261-270, 2002.

[36] T. Downing, *Java RMI: Remote Method Invocation*, IDG Books Worldwide, Foster City, CA, USA, 1998.

[37] W. Edwards, *Core Jini*, Prentice Hall PTR Upper Saddle River, NJ, USA,1999.

[38] F. Estievenart, A. Francois, J. Henrard and J. Hainaut, "A Tool-Supported Method to Extract

Data and Schema from Web Sites", *5th IEEE International Workshop on Web Site Evolution (WSE)*, pp 3-11,2003.

[39] J. Ferrante, K. Ottenstein and J. Warren, "The Program Dependence Graph and Its Use in Optimization", *ACM Transactions on Programming Languages and Systems*, Vol. 9, No. 3, 1987.

[40] B. Korel and J. Rilling, "Dynamic Program Slicing Methods", *Information and Software Technology*, Vol. 40, No. 11-12, pp. 647-659, 1998.

[41] I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing", *Lecture Notes in Computer Science*, Vol.2735, pp. 118-128, 2003.

[42] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[43] I. Foster, C. Kesselman, J. Nick and S. Tuecke, *Grid Computing*, John Wiley & Sons Ltd, 2003.

[44] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organisations", *International Journal of High Performance Computing Applications*, Vol. 15, No. 3, pp. 200-222, 2001.

[45] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 2003.

[46] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe and S. Weerawarana, *Modeling Stateful Resources with Web Services*, Globus Alliance, 2004.

[47] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, "The WS-Resources Framework, Version 1.0", *White Paper*, 2004. Available at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[48] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling and S. Tuecke, "Modeling and managing State in distributed systems: the role of OGSI and WSRF", *Proceedings of the IEEE*, Vol. 93, No. 3, pp. 604 – 612, 2005.

[49] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, "Web Services Resource Properties, Version 1.2", White Paper, 2005. Available at: http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-cd-01.pdf.

[50] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, "Web Services Service Group, Version 1.2", *White Paper*, 2005. Available at: http://docs.oasis-open.org/wsrf/wsrf-ws_ service_group-1.2-spec-cd-01.pdf.

[51] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, "Web Services Resource Lifetime, Version 1.2", White Paper, 2005. Available at: http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-cd-01.pdf.

[52] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, "Web Services Base Faults, Version 1.2", *White Paper*, 2005. Available at: http://docs.oasis-open.org/wsrf/wsrf-ws_ base_faults-1.2-spec-cd-01.pdf.

[53] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, "Web Services Notification, Version 1.2", *White Paper*, 2005. Available at: http://www.oasis-open.org/committees/tc_ home.php?wg_abbrev=wsn.

[54] I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling and P. Anderbilt, "Open Grid Services Infrastructure (OGSI) Version 1.0", *Global Grid Forum Draft*, Global Grid Forum Recommendation, 2003.

[55] G. Fox, "Data and Metadata on the Semantic Grid", *Computing in Science & Engineering*, Vol. 5, No. 5, pp. 76-78, 2003.

[56] N. Furmento, A. Mayer, S. McGough, S. Newhouse and J. Darlington, "A Component Framework for HPC Applications", *Lecture Notes in Computer Sciences*, Vol. 2150, pp. 540-548, 2001.

[57] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field and J. Darlington, "ICENI: Optimisation of Component Applications within a Grid Environment", *Journal of Parallel Computing*, Vol. 28, No. 12, pp. 1753-1772, 2002.

[58] K. Gallagher and J. Lyle, "Using Program Slicing in Software Maintenance", *IEEE Transactions on Software Engineering*, Vol. 17, No. 8, pp. 751-761, 1991.

[59] L. Gao, Y. Ding and L. Ren, "A Novel Ecological Network-Based Computation Platform as a Grid Middleware System", *International Journal of Intelligent Systems*, Vol. 19, No. 10, pp. 859-884, 2004.

[60] M. Geldof, *The Semantic Grid: Will Semantic Web and Grid go hand in hand*, European Commission DG Information Society Unit Grid Technologies, 2004.

[61] R. Gordon, *Essential JNI: Java Native Interface*, Prentice Hall PTR, ISBN: 0136798950, 1998.

[62] D. Gorissen, P. Wendykier, D. Kurzyniec and V. Sunderam, "Integrating Grid Information Services with JNDI", *15th International Heterogeneous Computing Workshop (HCW)*, Greece, 2006.

[63] R. Grimes and R. Grimes, *Professional DCOM Programming*, Wrox Press Ltd. Birmingham, UK, 1997.

[64] A. Grimshaw and A. Natrajan, "Legion: Lessons Learned Building a Grid Operating System", *Proceedings of the IEEE*, Vol. 93, No. 3, pp. 589-603, 2005.

[65] D. Gunter and K. Jackson, "The Applicability of RDF-Schema as a Syntax for Describing Grid Resource Metadata", *GWD-GIS-020-1*, Global Grid Forum, June 2001.

[66] G. Heineman and W. Councill, *Component Based Software Engineering : Putting the Pieces Together*, ACM Press, 2001.

[67] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*, Addison-Wesley Longman Publishing, Boston, MA, USA, 1999.

[68] S. Horwitz, T. Reps and D. Binkley, "Interprocedural Slicing Using Dependence Graphs", *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 1, pp. 26-60, 1990.

[69] N. Howarth, "Abstract Syntax Tree Design", *APM.1551*, APM Ltd., Cambridge, U.K, 1995.

[70] N. Howarth, "Rules for Type Inferencing", *APM.1552*, APM Ltd., Cambridge, U.K, 1995.

[71] Y. Huang, I. Taylor, D. Walker and R. Davies, "Wrapping Legacy Codes for Grid-Based Applications", *17th International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 139-145, 2003.

[72] M. Hull, P. Nicholl and Y. Bi, "Approaches to Component Technologies for Software Reuse of Legacy Systems", *Computing and Control Engineering Journal*, Vol. 12, No. 6, pp. 281-287, 2001.

[73] M. Humphrey, G. Wasson, M. Morgan and N. Beekwilder, "An Early Evaluation of WSRF and WS-Notification via WSRF.NET", *5th IEEE/ACM International Workshop on Grid Computing (GRID)*, pp. 172-181, 2004.

[74] M. Humphrey, G. Wasson, J. Gawor, J. Bester, S. Lang, I. Foster, S. Pickles and M. Keown, "State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations", *14th International Symposium on High Performance Distributed Computing (HPDC)*, pp. 3-13, 2005.

[75] J. Hunter, *Java Servlet Programming*, O'Reilly Publishing, ISBN 0596000405, 2001.

[76] P. Huy, K. Takahiro and H. Tetsuo, "Web Service Gateway - A Step Forward to E-Business",

IEEE International Conference on Web Services (ICWS), pp. 648-655, 2004.

[77] J. Hwang and P. Aravamudham, "Middleware Services for P2P Computing in Wireless Grid Networks", *IEEE Internet Computing*, Vol. 8, No. 4, pp. 40-46, 2004.

[78] *IEEE Std. 1219: Standard for Software Maintenance,* Los Alamitos CA., USA. IEEE Computer Society Press, 1993.

[79] *ISO12207*, International. Standards Organisation, Information Technology Software Lifecycle Processes, Geneva, Switzerland, 1995.

[80] B. Jacob, "Grid Computing: What Are the Key Components", *IBM DeveloperWorks*, 2003.

[81] S. Jean, V. Chua, P Echevarria and J. M. Mendoza, "Bayanihan Computing .NET: Grid Computing with XML Web Services", *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, Germany, 2002.

[82] Y. Jiang and E. Stroulia, "Towards reengineering Web sites to Web-services providers", *8th European Conference on Software Maintenance and Reengineering (CSMR)*, pp. 296-305, 2004.

[83] C. Johnson, "Basic research Skills in Computing Science", Department of Computer Science, Glasgow University, UK. Available at: http://www.dcs.gla.ac.uk/~johnson/teaching/ research_ skills/ basics.html.

[84] J. Joseph and C. Fellenstein, *Grid Computing*, IBM Press, ISBN: 0131456601, 2004.

[85] P. Kacsuk, A. Goyeneche, T. Delaitre, T. Kiss, Z. Farkas and T. Boczko, "High-Level Grid Application Environment to Use Legacy Codes as OGSA Grid Services", *5th IEEE/ACM International Workshop on Grid Computing (GRID)*, pp. 428-435, 2004.

[86] R. Koschke and T. Eisenbarth, "A Framework for Experimental Evaluation of Clustering Techniques", *8th International Workshop on Program Comprehension (IWPC)*, Limerick, Ireland, pp. 201-210, 2000.

[87] M. Krajecki, O. Flauzac and P. Merel, "Focus on the Communication Scheme in the Middleware CONFIIT using XML-RPC", *18th International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 2289-2294, 2004.

[88] D. Kuck, R. Kuhn, D. Padua, B. Leasure and M. Wolfe, "Dependence Graphs and Compiler Optimizations", *8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Virginia, pp. 207-218, 1981.

[89] J. Li and H. Yang, "Incorporating Legacy System into Semantic Grid Framework", *IEEE International Conference on Information Reuse and Integration (IRI)*, USA, 2006.

[90] J. Li and H. Yang, "Reengineering Websites into Stateful Resources for Grid Service Oriented Evolution", *International Journal of Multiagent and Grid Systems*, IOS Press, Vol. 2, No. 2, 2006.

[91] J. Li, Z. Zhang and H. Yang, "A Grid Oriented Approach to Reusing Legacy Code in ICENI Framework", *IEEE International Conference on Information Reuse and Integration (IRI)*, USA, 2005.

[92] J. Li, Z. Zhang, B. Qiao and H. Yang, "A Component Mining Approach to Incubate Grid Services in Object-Oriented Legacy Systems", *International Journal of Automation and Computing*, Vol. 3, No.1, 2006.

[93] X. Liu, H. Yang and H. Zedan, "On the Assessment of Re-engineering Tools", *International Workshop on Empirical Studies in Software Maintenance (WESS)*, Oxford, UK, 1999.

[94] Y. Maarek, R. Fagin, I. Shaul and D. Pelleg, "Ephemeral Document Clustering for Web Applications", *Technical Report RJ 10186*, IBM Research, 2000.

[95] F. Manola and E. Miller, "Rdf primer", *World Wide Web Consortium (W3C) Working Draft*, February 2004. Available at: http://www.w3.org/TR/ rdf-primer.

[96] S. Matsuoka, S. Shinjo, M. Aoyagi, S. Sekiguchi, H. Usami and K. Miura, "Japanese

Computational Grid Research Project: NAREGI", *Proceedings of the IEEE*, Vol. 93, No. 3, pp. 522-533, 2005.

[97] A. Mayer, S. McGough, M. Gulamali, L. Young, J. Stanton, S. Newhouse and J. Darlington, "Meaning and Behaviour in Grid Oriented Components", *3rd International Workshop on Grid Computing (GRID)*, pp. 100-111, 2002.

[98] D. Merrill, "Using Eclipse to Develop Grid Services", *IBM DeveloperWorks*, 2005.

[99] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler and S. Zhou, "Process Migration", *ACM Computing Surveys*, Vol. 32, No. 3, pp 241-284, 2000.

[100] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu, "Peer-to-Peer Computing", *Technical Report HPL-2002-57*, HP Laboratories, 2002.

[101] M. Mock, D. Atkinson and S. Eggers, "Program Slicing with Dynamic Points-to Sets", *IEEE Transactions on Software Engineering*, Vol. 31, No. 8, pp. 657-678, 2005.

[102] R. Monson-Haefel, *Enterprise JavaBeans*, O'Reilly Publishing, ISBN: 0130648841, 2001.

[103] J. Moreira, S. Midkiff, M. Gupta, P. Atrigas, P. Wu and G. Almasi, "The NINJA project: Making Java Work for High Performance Computing", *Communications of the ACM*, Vol. 44, No. 10, 2001.

[104] T. Mowbray and R. Zahavi, *The Essential CORBA: Systems Integration Using Distributed Objects*, John Wiley and Sons, ISBN: 0471106119, 1995.

[105] T. Myer, "Grid Computing: Conceptual Flyover for Developers", *IBM Developerworks*, 2003.

[106] C. Nester, M. Philippsenand and B. Haumacher, "A More Efficient RMI for Java", *ACM conference on Java Grande*, 1999.

[107] O. Nierstrasz, and D. Tsichritzis, *Object-Oriented Software Composition*, Englewood Cliffs, NJ: Prentice Hall, 1995.

[108] A. Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly & Associates, Sebastopol, CA, 2001.

[109] R. Orfali, D. Harkey, *Client/server programming with Java and CORBA*, John Wiley & Sons, NY, USA, 1998.

[110] A. Othman, P. Dew, K. Djemamem and I. Gourlay, "Adaptive Grid Resource Brokering", *IEEE International Conference on Cluster Computing*, pp.172-179, 2003.

[111] C. Pairot, P. Garcia, R. Mondejar and A. Skarmeta, "P2PCM: A Structured Peer-to-Peer Grid Component Model", *Lecture Notes in Computer Science*, Vol. 3516, pp. 246-249, 2005.

[112] T. Phan, L. Huang and C. Dulan, "Challenge: Integrating Mobile Wireless Devices into the Computational Grid", *8th Annual International Conference on Mobile Computing and Networking*, USA, pp. 271–278, 2002.

[113] R. Prieto-Diaz, "Domain analysis: An introduction", *ACM SigSoft Engineering Notes*, Vol. 15, No. 2, pp. 47-54, 1990.

[114] B. Qiao, *Evolution of Web-based Systems in Model Driven Architecture*, Ph.D Thesis, De Montfort University, 2005.

[115] B. Qiao, H. Yang, W. Chu and B. Xu, "Bridging Legacy Systems to Model Driven Architecture", *27th Annual International Computer Software and Application Conference (COMPSAC)*, USA, pp. 204- 309, 2003.

[116] F. Ricca and P. Tonella, "Using Clustering to Support the Migration from Static to Dynamic Web Pages", *11th IEEE International Workshop on Program Comprehension (IWPC)*, pp.207-216, 2003.

[117] H. Romesburg, *Cluster Analysis for Researchers*, Lulu Press, 2004.

[118] D. Roure, N. Jennings and N. Shadbolt., "The Semantic Grid: A Future E-Science Infrastructure", *UKeS-2002-02*, UK National e-Science Centre, UK, 2002.

[119] D. Roure, N. Jennings and N. Shadbolt, "The Semantic Grid: Past, Present, and Future," *Proceedings of the IEEE*, Vol. 93, No. 3, pp. 669-681, 2005.

[120] D. Roure, M. Baker, N. Jennings and N. Shadbolt, "*The Evolution of The Grid", In F. Berman, A. J. G. Hey, and G. Fox, editors, "Grid Computing: Making The Global Infrastructure a Reality"*, JohnWiley & Sons, pp. 65–100, 2003.

[121] S. Rugaber, "A Tool Suite for Evolving Legacy Software", *IEEE International Conference on Software Maintenance (ICSM)*, pp.33-39, 1999.

[122] R. Schwanke, "An Intelligent Tool for Re-engineering Software Modularity", *13th International Conference on Software Engineering (ICSE)*, USA, pp. 83-92, 1991.

[123] R. Seacord, D. Plakosh and G. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*, Addison-Wesley, ISBN: 0321118847, 2003.

[124] M. Serrano, D. Carver and C. Montes, "Reengineering of Legacy Systems to Distributed Object Environments", *Journal of Systems and Software*, Vol. 64, No. 1, pp. 37-55, 2002.

[125] R. Sessions, *COM and DCOM: Microsoft's Vision for Distributed Objects*, John Wiley & Sons, NY, USA, 1997.

[126] J. Siegel, S. Baker, M. Balick, D. Frantz, H. Mirsky et al., *COBRA Fundamentals and Programming*, John Wiley & Sons, NY, USA 1996.

[127] H. Sneed, "Object Oriented COBOL Recycling", *3rd IEEE Working Conference of Reverse Engineering (WCRE)*, USA, pp. 169-178, 1996.

[128] H. Sneed and S. Sneed, "Creating Web Services from Legacy Host Programs", *5th IEEE International Workshop on Web Site Evolution (WSE)*, pp. 59-69, 2003.

[129] H. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura and A. Chien, "The MicroGrid: A Scientific Tool for Modeling Computational Grids", *Scientific Programming*, IOS Press, Vol. 8, No. 3, pp. 127-141, 2000.

[130] B. Sotomayor, "The Globus Toolkit 4 Programmer's Tutorial", University of Chicago, Department of Computer Science. Available at: http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html.

[131] E. Stroulia, J. Thomson and G. Situ, "Constructing XML-Speaking Wrappers for Web Applications: Towards An Interoperating Web", *7th Working Conference on Reverse Engineering (WCRE)*, pp. 59-68, 2000.

[132] D. Sun, K. Wong, K and D. Moise, "Lessons Learned in Web Site Architectures for Public Utilities", *5th IEEE International Workshop on Web Site Evolution (WSE)*, pp. 93-100, 2003.

[133] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison Wesley Professional, 1997.

[134] D. Talia, "The Open Grid Services Architecture: Where the Grid Meets the Web", *IEEE Internet Computing*, Vol. 6, No. 6, pp. 67-71, 2002.

[135] J. Tang, W. Tong, J. Ding and L. Cai, "MOM-G Message-Oriented Middleware on Grid Environment Based on OGSA", *International Conference on Computer Networks and Mobile Computing (ICCNMC)*, pp. 424-427, 2003.

[136] F. Tao, S. Cox, L. Chen, N. Shadbolt, F. Xu, C. Puleston, C. Goble and W. Song, "Towards the Semantic Grid: Enriching Content for Management and Reuse", UK E-Science All Hands Meeting, pp. 695-702, 2003.

[137] P. Thiran and J. Hainaut, "Wrapper Development for Legacy Data Reuse", *8th IEEE Working Conference on Reverse Engineering*, Germany, pp 198-207, 2001.

[138] H. Tianfield and R. Unland, "Towards automatic computing system," *Journal of Engineering Applications of Artificial Intelligence, (EAAI)*, Elsevier. Vol. 17, No.7, pp. 689-699, 2004.

[139] F. Tip, "A Survey of Program Slicing Techniques", *Journal of Programming Languages*, Vol. 3, No. 3, pp. 121-189, 1995.

[140] P. Tonella, F. Ricca, E. Pianta and C. Girardi, "Using Keyword Extraction for Web Site Clustering", *5th IEEE International Workshop on Web Site Evolution (WSE)*, pp. 41-48, 2003.

[141] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, D. Snelling and P. Vanderbilt, *Open Grid Services Infrastructure (OGSI), Version 1.0*, Global Grid Forum, June, 2002. Available at: http://www.ggf.org.

[142] V. Turau. "Making Legacy Data Accessible for XML Applications", Available at: http://www.informatik.fh-wiesbaden.de/simturau/veroeff.html, 1999.

[143] V. Tzerpos and R. Holt, "MoJo: A Distance Metric for Software Clustering", *6th Working Conference on Reverse Engineering (WCRE)*, Atlanta, pp. 187-193, 1999.

[144] J. Unger and M. Haynos, "A Visual Tour of Open Grid Services Architecture", *IBM DeveloperWorks*, 2003.

[145] D. Wang, E. Carr, M. Palmer, M. Berry and L. Gross, "A Grid Service Module for Natural-Resource Managers", *IEEE Internet Computing*, Vol. 9, No. 1, pp. 35-41, 2005.

[146] J. Ward, "Hierarchical Grouping to Optimize an Objective Function", *Journal of the American Statistical Association*, Vol. 58, No. 301, pp. 236-244, 1963.

[147] G. Wasson and M. Humphrey, "Exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments", *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.

[148] M. Weiser, "Program Slicing", *IEEE Transactions on Software Engineering*, Vol. 10, No. 4, 1984.

[149] T. Wiggerts, "Using Clustering Algorithms in Legacy Systems Remodularization", *4th Working Conference on Reverse Engineering (WCRE)*, Netherlands, 1997.

[150] H. Yang, X. Liu and H. Zedan, "Abstraction: A Key Notion for Reverse Engineering in A

System Re-engineering Approach", *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 12, No. 5, pp. 197-228, 2000.

[151] H. Yang and M. Ward, *Successful Evolution of Software Systems*, Artech House, Boston, USA & London, UK, 2003.

[152] A. Yeh, D. Harris and H. Reubenstein, "Recovering Abstract Data Type and Object Instances from a Conventional Procedural Language", *Working Conference on Reverse Engineering (WCRE)*, pp. 227-236, 1995.

[153] Y. Zou, *Migration of Procedural Systems to Object Oriented Platforms*, Ph.D Thesis, University of Waterloo, Department of Electrical and Computer Engineering, 2003.

[154] Z. Zhang and H. Yang, "Incubating Services in Legacy Systems for Architectural Migration", *11th IEEE Asia-Pacific Software Engineering Conference (APSEC)*, pp. 196-203, 2004.

[155] S. Zweben, S. Edwards, B. Weide and J. Hollingsworth, "The Effects of Layering and Encapsulation on Software Development Cost and Quality", *IEEE Transactions on Software Engineering*, Vol. 21, No. 3, pp. 200-208, 1995.

Appendix A

Source Code of Example Legacy Bank System

This is an automated software system for Bank Management which can handle accounts of customers. It uses files to handle the daily transactions, account management and user management. This Bank Management System performs the following functions: create individual accounts, manage existing accounts and view daily transactions functions.

Its not a complete accounting software just like implemented in the banks, but still it can manage the accounts of the customers using the files at backend. It has a nice graphical layout written in C Language.

This software is provided by MYCPLUS for academic purpose with the source code which is free to use, to modify and to be changed as requirements.

```
*/
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<malloc.h>
#include<dos.h>
#include<fstream.h>
#include<stdlib.h>
#include<string.h>
void intro(void);
int main_menu(void);
void new_account(void);
void new_account_gui(void);
int getaccount_no(char *);
void sav_account(int,char*,int,char*,int,int,int,int,int);
void update_main(char *);
void process(void);
void list_all(void);
```

```
void print_it(char*);
void list_all_gui(void);
void trans_menu(void);
int trans_gui(void);
void show_trans(void);
void mod_trans(void);
void see_it(void);
void report(void);
void dep_gui(void);
void deposite(void);
void mod_choice(void);
void withdraw(void);
main() {
        intro();
        process();
        return 0;
}
void process(void) {
        int option;
        option=main_menu();
        if(option==1)
new_account();
if(option==2)
        list_all();
        if(option==3)
                trans_menu();
                if(option==4)
                        exit(0);
                }
                void intro(void) {
                        void *image;
                        int size;
                        int dr=9,mode=2;
                        initgraph(&dr,&mode,"..\\bgi");
                        size=imagesize(140,140,500,250);
                image=malloc(size);
                setfillstyle(SOLID_FILL,GREEN);
                circle(200,200,50);
                floodfill(200,200,WHITE);
                setcolor(LIGHTGRAY);
                circle(200,200,19);
```

```
setcolor(WHITE);
circle(200,200,18);
circle(200,200,49);
circle(200,200,53);
circle(200,200,20);

arc(215,215,350,90,30);
arc(210,182,90,194,30);
arc(180,195,180,300,30);
settextstyle(1,HORIZ_DIR,1);
setcolor(LIGHTGRAY);
outtextxy(270,180,"National Bank");

outtextxy(270,185,"_____");
outtextxy(270,210,"Of Pakistan Pvt Ltd.");
settextstyle(0,HORIZ_DIR,0);
setcolor(DARKGRAY);
outtextxy(270,240,"COPYRIGHT 2002");
getimage(141,141,499,259,image);
cleardevice();
for(int count=1;count<300;count+=2)
      putimage(1+count,100,image,COPY_PUT);
      for(int down=0;down<640;++down) {
            delay(5);
            line(1,220,1+down,220);
      }
      setcolor(BLUE);
      for(int bottom=0;bottom<300;bottom+=4)
      line(1,220+bottom,640,220+bottom);
      free(image);
      getch();
      closegraph();
}
// ---------- m a i n - m e n u ----------
int main_menu(void) {
int option;
int size;
int dr=9,mode=2;
initgraph(&dr,&mode,"..\\bgi");
void *main_window,*button_down;
size=imagesize(10,70,251,351);
```

```
main_window=malloc(size);
size=imagesize(10,310,251,351);
button_down=malloc(size);
setfillstyle(SOLID_FILL,LIGHTGRAY);
setcolor(DARKGRAY);
rectangle(1,1,640,480);
floodfill(3,3,DARKGRAY);
setcolor(WHITE);
line(1,1,1,480);
line(1,1,640,1);
setcolor(BLUE);
rectangle(3,3,637,15);
setfillstyle(SOLID_FILL,BLUE);
floodfill(5,5,BLUE);
setcolor(DARKGRAY);
rectangle(620,4,635,14);
rectangle(602,4,617,14);
rectangle(585,4,599,14);
setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(586,6,DARKGRAY);
floodfill(604,6,DARKGRAY);
floodfill(624,6,DARKGRAY);
setcolor(BLACK);
line(622,6,633,12);
line(633,6,622,12);
rectangle(604,6,615,12);
line(587,12,597,12);
setcolor(WHITE);
line(620,4,620,14);
line(620,4,635,4);
line(602,4,617,4);
line(602,4,602,14);
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
```

```
setcolor(DARKGRAY);
outtextxy(40,40,);
rectangle(10,70,200,100);
setcolor(WHITE);
line(9,69,9,99);
line(10,69,200,69);
setcolor(BLACK);
outtextxy(30,80,"M A I N      M E N U");
setcolor(DARKGRAY);
rectangle(10,110,250,150);
setcolor(BLACK);
rectangle(10,110,251,151);
setcolor(WHITE);
line(10,110,10,150);
line(10,110,250,110);
setcolor(BLACK);
outtextxy(40,130,"1.    Create new account");
setcolor(DARKGRAY);
rectangle(10,160,250,200);
setcolor(BLACK);
rectangle(10,160,251,201);
setcolor(WHITE);
line(10,160,10,200);
line(10,160,250,160);
setcolor(BLACK);
outtextxy(40,180,"2.    List all accounts");
setcolor(DARKGRAY);
rectangle(10,210,250,250);
setcolor(BLACK);
rectangle(10,210,251,251);
setcolor(WHITE);
line(10,210,10,250);
line(10,210,250,210);
setcolor(BLACK);
outtextxy(40,230,"3.    Show individual info");
setcolor(DARKGRAY);
rectangle(10,260,250,300);
setcolor(BLACK);
rectangle(10,260,251,301);
setcolor(WHITE);
line(10,260,10,300);
```

```
line(10,260,250,260);
setcolor(BLACK);
outtextxy(40,280,"4.  Quit");
setcolor(LIGHTGRAY);
rectangle(10,310,250,350);
getimage(10,70,251,351,main_window);
setcolor(WHITE);
rectangle(10,310,251,351);
setcolor(DARKGRAY);
line(10,310,10,350);
line(11,311,11,350);
line(10,310,250,310);
line(11,311,250,311);
setcolor(BLACK);
getimage(10,310,251,351,button_down);
cleardevice();
setfillstyle(SOLID_FILL,LIGHTGRAY);
setcolor(DARKGRAY);
rectangle(1,1,640,480);
floodfill(3,3,DARKGRAY);
setcolor(WHITE);
line(1,1,1,480);
line(1,1,640,1);
setcolor(BLUE);
rectangle(3,3,637,15);
setfillstyle(SOLID_FILL,BLUE);
floodfill(5,5,BLUE);
setcolor(DARKGRAY);
rectangle(620,4,635,14);
rectangle(602,4,617,14);
rectangle(585,4,599,14);
setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(586,6,DARKGRAY);
floodfill(604,6,DARKGRAY);
floodfill(624,6,DARKGRAY);
setcolor(BLACK);
line(622,6,633,12);
line(633,6,622,12);
rectangle(604,6,615,12);
line(587,12,597,12);
setcolor(WHITE);
```

```
line(620,4,620,14);
line(620,4,635,4);
line(602,4,617,4);
line(602,4,602,14);
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
putimage(10,70,main_window,COPY_PUT);
char check;
setcolor(BLACK);
check=getch();
if(check==49) {
      putimage(10,110,button_down,COPY_PUT);
      outtextxy(40,130,"1.   Create new account");
      option=1;
      delay(600);
}
if(check==50) {
      putimage(10,160,button_down,COPY_PUT);
      outtextxy(40,180,"2.   List all accounts");
      option=2;
      delay(600);
}
if(check==51) {
      putimage(10,210,button_down,COPY_PUT);
      outtextxy(40,230,"3.   Show individual info");
      option=3;
      delay(600);
}
if(check==52) {
      putimage(10,260,button_down,COPY_PUT);
      outtextxy(40,280,"4.   Quit");
      option=4;
```

```
        delay(600);
}
free(main_window);
free(button_down);
cleardevice();
closegraph();
return(option);
}
void new_account(void) {
        int withdraw=0;
        int deposit=0;
        int account_no=888;
        int balance=0;
        char address[25];
        char name[10]={'D','E','F','A','U','L','T'};
        struct date dat;
        getdate(&dat);
        account_no=getaccount_no(name);
        int year=dat.da_year;
        int day=dat.da_day;
        int month=dat.da_mon;
        --account_no;
        new_account_gui();
        gotoxy(20,9);
        cout<<account_no;
        gotoxy(42,9);
        cout<<day<<" of "<<month<<" "<<year;
        gotoxy(40,13);
        cout<<" ";
        cin>>name;
        gotoxy(40,17);
        cout<<" ";
        cin>>balance;
        gotoxy(40,21);
        cout<<" ";
        cin>>address;

        sav_account(account_no,name,balance,address,day,month,year,deposit,withdraw
        );
        cleardevice();
        closegraph();
```

```
        process();
}
int getaccount_no(char *name) {
        char string[10];
        int count=0;
        int account_no=0;
        ifstream pfile("main");
if(pfile==NULL) {
        account_no=2;
        pfile.close();
        ofstream pfile2("main");
        pfile2<<name;
        pfile2.close();
        return account_no;
}
ifstream pfile3("main");
while(string[0]!=NULL) {
        pfile3>>string;
        ++count;
}
pfile3.close();
        account_no=count;
        return account_no;
}
 void sav_account(int account_no,char *name,int balance,char *address,int day,int
 month,int year,int deposit,int withdraw) {
        ofstream pfile(name);
        pfile<<account_no;
        pfile<<endl;
        pfile<<name;
        pfile<<endl;
        pfile<<balance;
        pfile<<endl;
        pfile<<address;
        pfile<<endl;
        pfile<<day<<"of"<<month<<"-"<<year;
        pfile<<endl;
        pfile<<deposit;
        pfile<<endl;
        pfile<<withdraw;
        pfile.close();
```

```
        update_main(name);
}
void update_main(char *name) {
        ofstream pfile;
        pfile.open("main",ios::app);
                pfile<<endl;
                pfile<<name;
        pfile.close();
}
void new_account_gui(void) {
        int dr=9,mode=2;
        initgraph(&dr,&mode,"..\\bgi");
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        setcolor(DARKGRAY);
        rectangle(1,1,640,480);
        floodfill(3,3,DARKGRAY);
        setcolor(WHITE);
        line(1,1,1,480);
        line(1,1,640,1);
        setcolor(BLUE);
        rectangle(3,3,637,15);
        setfillstyle(SOLID_FILL,BLUE);
        floodfill(5,5,BLUE);
        setcolor(DARKGRAY);
        rectangle(620,4,635,14);
        rectangle(602,4,617,14);
        rectangle(585,4,599,14);
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        floodfill(586,6,DARKGRAY);
        floodfill(604,6,DARKGRAY);
        floodfill(624,6,DARKGRAY);
        setcolor(BLACK);
        line(622,6,633,12);
        line(633,6,622,12);
        rectangle(604,6,615,12);
        line(587,12,597,12);
        setcolor(WHITE);
        line(620,4,620,14);
        line(620,4,635,4);
        line(602,4,617,4);
        line(602,4,602,14);
```

```
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
rectangle(10,70,200,100);
outtextxy(20,80,"N E W   A C C O U N T");
setcolor(WHITE);
line(9,70,9,100);
line(10,69,200,69);
setcolor(DARKGRAY);
rectangle(10,110,500,400);
setcolor(WHITE);
line(11,111,11,399);
line(11,111,499,111);
setcolor(DARKGRAY);
outtextxy(40,130,"ACCOUNT NO DATED ");
setcolor(DARKGRAY);
rectangle(130,125,200,150);
rectangle(310,125,450,150);
setcolor(WHITE);
line(199,149,131,149);
line(199,149,199,126);
line(449,149,311,149);
line(449,149,449,126);
setcolor(DARKGRAY);
line(130,125,130,150);
line(130,125,200,125);
line(310,125,310,150);
line(310,125,450,125);

line(30,160,470,160);
setcolor(WHITE);
line(30,161,470,161);
setcolor(DARKGRAY);
```

```
outtextxy(30,200,"Enter your name (10 char max)");
outtextxy(30,270,"Enter your balance (1000 min)");
outtextxy(30,340,"Enter your address (25 char max)");
setcolor(WHITE);
rectangle(300,190,470,210);
setfillstyle(SOLID_FILL,WHITE);
floodfill(310,195,WHITE);
line(472,212,472,189);
line(472,212,299,212);
setcolor(DARKGRAY);
line(300,190,300,210);
line(300,190,470,190);
line(299,189,299,211);
line(299,189,471,189);
setcolor(WHITE);
rectangle(300,260,470,280);
setfillstyle(SOLID_FILL,WHITE);
floodfill(310,265,WHITE);
line(472,282,472,259);
line(472,282,299,282);
setcolor(DARKGRAY);
line(300,260,300,280);
line(300,260,470,260);
line(299,259,299,281);
line(299,259,471,259);
setcolor(WHITE);
rectangle(300,330,470,350);
setfillstyle(SOLID_FILL,WHITE);
floodfill(310,335,WHITE);
line(472,352,472,329);
line(472,352,299,352);
setcolor(DARKGRAY);
line(300,330,300,350);
line(300,330,470,330);
line(299,329,299,351);
line(299,329,471,329);
}
void list_all(void) {
    struct date d;
    getdate(&d);
    list_all_gui();
```

```
ifstream pfile("main");
int count=0;
char string[10];
while(string[0]!=NULL) {
        pfile>>string;
        ++count;
}
--count;
pfile.close();
ifstream pfile2("main");
for(int var=count;var>0;--var) {
        pfile2>>string;
        if((var!=count)&&(string!=NULL))
                print_it(string);
        }
        pfile2.close();

        gotoxy(80,25);
        getch();
        cleardevice();
        closegraph();
        process();
}
void print_it(char *name) {
        static value=0;
        char address[25];
        char account_no[5];
        char balance[5];
        char withdraw[30];
        char date[20];
        char deposit[30];
        ifstream pfile(name);
        pfile>>account_no;
        pfile>>name;
        pfile>>balance;
        pfile>>address;
        pfile>>date;
        pfile>>deposit;
        pfile>>withdraw;
        pfile.close();
        gotoxy(6,10+value);
```

```
        cout<<"#";
        gotoxy(10,value+10);
        cout<<account_no;
        gotoxy(14,value+10);
        cout<<name;
        gotoxy(25,value+10);
        cout<<balance;
        gotoxy(35,value+10);
        cout<<address;
        gotoxy(50,value+10);
        cout<<date;
        ++value;
}
void list_all_gui(void) {
        int dr=9,mode=2;
        initgraph(&dr,&mode,"..\\bgi");
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        setcolor(DARKGRAY);
        rectangle(1,1,640,480);
        floodfill(3,3,DARKGRAY);
        setcolor(WHITE);
        line(1,1,1,480);
        line(1,1,640,1);
        setcolor(BLUE);
        rectangle(3,3,637,15);
        setfillstyle(SOLID_FILL,BLUE);
        floodfill(5,5,BLUE);
        setcolor(DARKGRAY);
        rectangle(620,4,635,14);
        rectangle(602,4,617,14);
        rectangle(585,4,599,14);
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        floodfill(586,6,DARKGRAY);
        floodfill(604,6,DARKGRAY);
        floodfill(624,6,DARKGRAY);
        setcolor(BLACK);
        line(622,6,633,12);
        line(633,6,622,12);
        rectangle(604,6,615,12);
        line(587,12,597,12);
        setcolor(WHITE);
```

```
line(620,4,620,14);
line(620,4,635,4);
line(602,4,617,4);
line(602,4,602,14);
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
rectangle(10,70,300,100);
setcolor(WHITE);
line(9,69,9,100);
line(10,69,300,69);
setcolor(DARKGRAY);
outtextxy(30,80,"A C C O U N T   L I S T I N G S");
      setcolor(WHITE);
      rectangle(30,110,610,450);
      setfillstyle(SOLID_FILL,WHITE);
      floodfill(35,115,WHITE);
      line(612,452,612,109);
      line(612,452,29,452);
      setcolor(DARKGRAY);
      line(30,110,30,450);
      line(30,110,610,110);
      line(29,110,29,450);
      line(29,109,610,109);
}
void trans_menu(void) {
      int check;
check=trans_gui();

if(check==1) {
      show_trans();
      see_it();
}
```

```
if(check==2) {
        mod_choice();
}
getch();
cleardevice();
closegraph();
process();
}
int trans_gui(void) {
        int dr=9,mode=2,option;
        initgraph(&dr,&mode,"..\\bgi");
        void *main_menu,*button_down;
        int size;
        size=imagesize(10,110,251,201);
        main_menu=malloc(size);
        size=imagesize(10,310,251,351);
        button_down=malloc(size);
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        setcolor(DARKGRAY);
        rectangle(1,1,640,480);
        floodfill(3,3,DARKGRAY);
        setcolor(WHITE);
        line(1,1,1,480);
        line(1,1,640,1);
        setcolor(BLUE);
        rectangle(3,3,637,15);
        setfillstyle(SOLID_FILL,BLUE);
        floodfill(5,5,BLUE);
        setcolor(DARKGRAY);
        rectangle(620,4,635,14);
        rectangle(602,4,617,14);
        rectangle(585,4,599,14);
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        floodfill(586,6,DARKGRAY);
        floodfill(604,6,DARKGRAY);
        floodfill(624,6,DARKGRAY);
        setcolor(BLACK);
        line(622,6,633,12);
        line(633,6,622,12);
        rectangle(604,6,615,12);
        line(587,12,597,12);
```

```
setcolor(WHITE);
line(620,4,620,14);
line(620,4,635,4);
line(602,4,617,4);
line(602,4,602,14);
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
rectangle(10,70,300,100);
setcolor(WHITE);
line(9,69,9,100);
line(10,69,300,69);
setcolor(DARKGRAY);
outtextxy(30,80," ACCOUNT TRANSACTION MENU");
setcolor(DARKGRAY);
rectangle(10,110,250,150);
setcolor(BLACK);
rectangle(10,110,251,151);
setcolor(WHITE);
line(10,110,10,150);
line(10,110,250,110);
setcolor(BLACK);
outtextxy(40,130,"1. View transaction");
setcolor(DARKGRAY);
rectangle(10,160,250,200);
setcolor(BLACK);
rectangle(10,160,251,201);
setcolor(WHITE);
line(10,160,10,200);
line(10,160,250,160);
setcolor(BLACK);
outtextxy(40,180,"2. Modify transaction");
getimage(10,110,251,201,main_menu);
```

```
setcolor(WHITE);
rectangle(10,310,251,351);
setcolor(DARKGRAY);
line(10,310,10,350);
line(11,311,11,350);
line(10,310,250,310);
line(11,311,250,311);
setcolor(BLACK);
getimage(10,310,251,351,button_down);
rectangle(10,310,251,351);
setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(15,315,BLACK);

setcolor(LIGHTGRAY);
rectangle(10,310,251,351);
setcolor(BLACK);
char check;
check=getch();
if(check==49) {
      putimage(10,110,main_menu,COPY_PUT);
      putimage(10,110,button_down,COPY_PUT);
      outtextxy(40,130,"1. View transaction");
      option=1;
      delay(600);
}
if(check==50) {
      putimage(10,110,main_menu,COPY_PUT);
      putimage(10,160,button_down,COPY_PUT);
      outtextxy(40,180,"2. Modify transaction");
      option=2;
      delay(600);
}
free(main_menu);
free(button_down);
      cleardevice();
      closegraph();
      return(option);
}
void show_trans(void) {
      int dr=9,mode=2;
```

```
initgraph(&dr,&mode,"..\\bgi");
setfillstyle(SOLID_FILL,LIGHTGRAY);
setcolor(DARKGRAY);
rectangle(1,1,640,480);
floodfill(3,3,DARKGRAY);
setcolor(WHITE);
line(1,1,1,480);
line(1,1,640,1);
setcolor(BLUE);
rectangle(3,3,637,15);
setfillstyle(SOLID_FILL,BLUE);
floodfill(5,5,BLUE);
setcolor(DARKGRAY);
rectangle(620,4,635,14);
rectangle(602,4,617,14);
rectangle(585,4,599,14);
setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(586,6,DARKGRAY);
floodfill(604,6,DARKGRAY);
floodfill(624,6,DARKGRAY);
setcolor(BLACK);
line(622,6,633,12);
line(633,6,622,12);
rectangle(604,6,615,12);
line(587,12,597,12);
setcolor(WHITE);
line(620,4,620,14);
line(620,4,635,4);
line(602,4,617,4);
line(602,4,602,14);
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
```

```
rectangle(10,70,200,100);
outtextxy(20,80,"SHOW TRANSACTION");
setcolor(WHITE);
line(9,70,9,100);
line(10,69,200,69);
setcolor(DARKGRAY);
line(30,170,610,170);
setcolor(WHITE);
line(30,171,610,171);
setcolor(DARKGRAY);
outtextxy(30,200,"Enter your account holder's name");
setcolor(WHITE);
rectangle(300,190,470,210);
setfillstyle(SOLID_FILL,WHITE);
floodfill(310,195,WHITE);
line(472,212,472,189);
line(472,212,299,212);
setcolor(DARKGRAY);
line(300,190,300,210);
line(300,190,470,190);
line(299,189,299,211);
        line(299,189,471,189);
}
void mod_trans(void) {
        int dr=9,mode=2;
        initgraph(&dr,&mode,"..\\bgi");
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        setcolor(DARKGRAY);
        rectangle(1,1,640,480);
        floodfill(3,3,DARKGRAY);
        setcolor(WHITE);
        line(1,1,1,480);
        line(1,1,640,1);
        setcolor(BLUE);
        rectangle(3,3,637,15);
        setfillstyle(SOLID_FILL,BLUE);
        floodfill(5,5,BLUE);
        setcolor(DARKGRAY);
        rectangle(620,4,635,14);
        rectangle(602,4,617,14);
        rectangle(585,4,599,14);
```

```
setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(586,6,DARKGRAY);
floodfill(604,6,DARKGRAY);
floodfill(624,6,DARKGRAY);
setcolor(BLACK);
line(622,6,633,12);
line(633,6,622,12);
rectangle(604,6,615,12);
line(587,12,597,12);
setcolor(WHITE);
line(620,4,620,14);
line(620,4,635,4);
line(602,4,617,4);
line(602,4,602,14);
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
rectangle(10,70,200,100);
outtextxy(20,80,"MODIFY ACCOUNT");
setcolor(WHITE);
line(9,70,9,100);
line(10,69,200,69);
setcolor(DARKGRAY);
line(30,170,610,170);
setcolor(WHITE);
line(30,171,610,171);
setcolor(DARKGRAY);
outtextxy(30,200,"Enter your account holder's name");
setcolor(WHITE);
rectangle(300,190,470,210);
setfillstyle(SOLID_FILL,WHITE);
floodfill(310,195,WHITE);
line(472,212,472,189);
```

```
        line(472,212,299,212);
        setcolor(DARKGRAY);
        line(300,190,300,210);
        line(300,190,470,190);
        line(299,189,299,211);
        line(299,189,471,189);
}
void see_it(void) {
        char name[10];
        char address[25];
        char account_no[5];
        char balance[5];
        char withdraw[30];
        char date[20];
        char deposit[30];
        gotoxy(40,13);
        cout<<" ";
        cin>>name;
        ifstream pfile(name);
        pfile>>account_no;
        pfile>>name;
        pfile>>balance;
        pfile>>address;
        pfile>>date;
        pfile>>deposit;
        pfile>>withdraw;
        pfile.close();
        report();
        gotoxy(20,10);
        cout<<"ACCOUNT HOLDER NAME: "<<name<<endl;
        gotoxy(20,11);
        cout<<"DATE OF ISSUE: "<<date<<endl;
        gotoxy(20,12);
        cout<<"BALANCE LEFT: "<<balance<<endl;
        gotoxy(20,13);
        cout<<"ADDRESS: "<<address<<endl;
        cout<<endl;
        gotoxy(20,15);
        cout<<"TRANSACTION REPORT"<<endl;
        gotoxy(20,16);
        cout<<"DEPOSIT REPORT: "<<deposit<<endl;
```

```
        gotoxy(20,17);
        cout<<"WITH DRAW REPORT: "<<withdraw<<endl;
}
void report(void) {
        int dr=9,mode=2;
        initgraph(&dr,&mode,"..\\bgi");
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        setcolor(DARKGRAY);
        rectangle(1,1,640,480);
        floodfill(3,3,DARKGRAY);
        setcolor(WHITE);
        line(1,1,1,480);
        line(1,1,640,1);
        setcolor(BLUE);
        rectangle(3,3,637,15);
        setfillstyle(SOLID_FILL,BLUE);
        floodfill(5,5,BLUE);
        setcolor(DARKGRAY);
        rectangle(620,4,635,14);
        rectangle(602,4,617,14);
        rectangle(585,4,599,14);
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        floodfill(586,6,DARKGRAY);
        floodfill(604,6,DARKGRAY);
        floodfill(624,6,DARKGRAY);
        setcolor(BLACK);
        line(622,6,633,12);
        line(633,6,622,12);
        rectangle(604,6,615,12);
        line(587,12,597,12);
        setcolor(WHITE);
        line(620,4,620,14);
        line(620,4,635,4);
        line(602,4,617,4);
        line(602,4,602,14);
        line(584,4,599,4);
        line(584,4,584,14);
        outtextxy(8,5,"BANKING SYSTEM");
        setcolor(WHITE);
        line(5,30,635,30);
        line(5,60,635,60);
```

```
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
rectangle(10,70,300,100);
setcolor(WHITE);
line(9,69,9,100);
line(10,69,300,69);
setcolor(DARKGRAY);
outtextxy(30,80,"TRANSACTION REPORT");
setcolor(WHITE);
rectangle(30,110,610,450);
setfillstyle(SOLID_FILL,WHITE);
floodfill(35,115,WHITE);
line(612,452,612,109);
line(612,452,29,452);
setcolor(DARKGRAY);
line(30,110,30,450);
line(30,110,610,110);
line(29,110,29,450);
      line(29,109,610,109);
}
void deposite(void) {
      char name[10];
      gotoxy(40,13);
      cout<<" ";
      cin>>name;
      int val_dep;
      char tmp_dep[5];
      char address[25];
      char dat[10];
      char deposit[30];
      char withdraw[40];
      int   account_no;
int balance;
ifstream pfile(name);
pfile>>account_no;
pfile>>name;
pfile>>balance;
pfile>>address;
```

```
pfile>>dat;
pfile>>deposit;
pfile>>withdraw;
dep_gui();
gotoxy(5,8);

cout<<account_no<<name<<balance<<dat<<deposit<<address<
<withdraw<<endl;
pfile.close();
strcat(deposit,"#");
gotoxy(40,13);
cout<<" ";
cin>>tmp_dep;
strcat(deposit,tmp_dep);
val_dep=atoi(tmp_dep);
balance+=val_dep;
ofstream pfile2(name);
pfile2<<account_no<<endl;
pfile2<<name<<endl;
pfile2<<balance<<endl;
pfile2<<address<<endl;
        pfile2<<dat<<endl;
        pfile2<<deposit<<endl;
        pfile2<<withdraw;
        pfile2.close();
        cleardevice();
        closegraph();
        process();
}
void dep_gui(void) {
        int dr=9,mode=2;
        initgraph(&dr,&mode,"..\\bgi");
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        setcolor(DARKGRAY);
        rectangle(1,1,640,480);
        floodfill(3,3,DARKGRAY);
        setcolor(WHITE);
        line(1,1,1,480);
        line(1,1,640,1);
        setcolor(BLUE);
        rectangle(3,3,637,15);
```

```
setfillstyle(SOLID_FILL,BLUE);
floodfill(5,5,BLUE);
setcolor(DARKGRAY);
rectangle(620,4,635,14);
rectangle(602,4,617,14);
rectangle(585,4,599,14);
setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(586,6,DARKGRAY);
floodfill(604,6,DARKGRAY);
floodfill(624,6,DARKGRAY);
setcolor(BLACK);
line(622,6,633,12);
line(633,6,622,12);
rectangle(604,6,615,12);
line(587,12,597,12);
setcolor(WHITE);
line(620,4,620,14);
line(620,4,635,4);
line(602,4,617,4);
line(602,4,602,14);
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
rectangle(10,70,200,100);
outtextxy(20,80,"MODIFY ACCOUNT");
setcolor(WHITE);
rectangle(30,110,610,130);
setfillstyle(SOLID_FILL,WHITE);
floodfill(310,115,WHITE);
line(612,132,612,109);
line(612,132,29,132);
setcolor(DARKGRAY);
line(30,110,30,130);
```

```
line(30,110,610,110);
line(29,109,29,131);
line(29,109,611,109);
setcolor(WHITE);
line(9,70,9,100);
line(10,69,200,69);
setcolor(DARKGRAY);
line(30,170,610,170);
setcolor(WHITE);
line(30,171,610,171);
setcolor(DARKGRAY);
outtextxy(30,200,"Enter the amount to deposite");
setcolor(WHITE);
rectangle(300,190,470,210);
setfillstyle(SOLID_FILL,WHITE);
floodfill(310,195,WHITE);
line(472,212,472,189);
line(472,212,299,212);
setcolor(DARKGRAY);
line(300,190,300,210);
line(300,190,470,190);
        line(299,189,299,211);
        line(299,189,471,189);
}
int mod_menu(void) {
        int dr=9,mode=2;
        initgraph(&dr,&mode,"..\\bgi");
        void *main_menu,*button_down;
        int size;
        size=imagesize(10,110,251,201);
        main_menu=malloc(size);
        size=imagesize(10,310,251,351);
        button_down=malloc(size);
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        setcolor(DARKGRAY);
        rectangle(1,1,640,480);
        floodfill(3,3,DARKGRAY);
        setcolor(WHITE);
        line(1,1,1,480);
        line(1,1,640,1);
        setcolor(BLUE);
```

```
rectangle(3,3,637,15);
setfillstyle(SOLID_FILL,BLUE);
floodfill(5,5,BLUE);
setcolor(DARKGRAY);
rectangle(620,4,635,14);
rectangle(602,4,617,14);
rectangle(585,4,599,14);
setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(586,6,DARKGRAY );
floodfill(604,6,DARKGRAY);
floodfill(624,6,DARKGRAY);
setcolor(BLACK);
line(622,6,633,12);
line(633,6,622,12);
rectangle(604,6,615,12);
line(587,12,597,12);
setcolor(WHITE);
line(620,4,620,14);
line(620,4,635,4);
line(602,4,617,4);
line(602,4,602,14);
line(584,4,599,4);
line(584,4,584,14);
outtextxy(8,5,"BANKING SYSTEM");
setcolor(WHITE);
line(5,30,635,30);
line(5,60,635,60);
setcolor(DARKGRAY);
line(5,29,635,29);
line(5,59,635,59);
setcolor(DARKGRAY);
outtextxy(40,40,);
rectangle(10,70,300,100);
setcolor(WHITE);
line(9,69,9,100);
line(10,69,300,69);
setcolor(DARKGRAY);
outtextxy(30,80,"MODIFY ACCOUNT");
setcolor(DARKGRAY);
rectangle(10,110,250,150);
setcolor(BLACK);
```

```
rectangle(10,110,251,151);
setcolor(WHITE);
line(10,110,10,150);
line(10,110,250,110);
setcolor(BLACK);
outtextxy(40,130,"1. Want to deposite");
setcolor(DARKGRAY);
rectangle(10,160,250,200);
setcolor(BLACK);
rectangle(10,160,251,201);
setcolor(WHITE);
line(10,160,10,200);
line(10,160,250,160);
setcolor(BLACK);
outtextxy(40,180,"2. Want to withdraw");
getimage(10,110,251,201,main_menu);

setcolor(WHITE);
rectangle(10,310,251,351);
setcolor(DARKGRAY);
line(10,310,10,350);
line(11,311,11,350);
line(10,310,250,310);
line(11,311,250,311);
setcolor(BLACK);
getimage(10,310,251,351,button_down);
rectangle(10,310,251,351);
setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(15,315,BLACK);
int option;
setcolor(LIGHTGRAY);
rectangle(10,310,251,351);
setcolor(BLACK);
char check;
check=getch();
if(check==49)
    {
        putimage(10,110,main_menu,COPY_PUT);
        putimage(10,110,button_down,COPY_PUT);
        outtextxy(40,130,"1. Want to deposite");
        option=1;
```

```
        delay(600);
}
if(check==50)
        {
                putimage(10,110,main_menu,COPY_PUT);
                putimage(10,160,button_down,COPY_PUT);
                outtextxy(40,180,"2. Want to withdraw");
                option=2;
                delay(600);
        }
        free(main_menu);
        free(button_down);
        return(option);
}
void mod_choice(void)
{
        int check;
        check=mod_menu();
        if(check==1)
                {
                        mod_trans();
        deposite();
}
if(check==2)
        {
                mod_trans();
                withdraw();
        }
}
void withdraw(void){
        int tmp;
        char name[10];
        char tmp_with[10];
        char address[25];
        char dat[10];
        char deposit[40];
        char withdraw[40];
        int   account_no;
        int balance;
        gotoxy(40,13);
        cout<<" ";
```

```
cin>>name;
ifstream pfile(name);
pfile>>account_no;
pfile>>name;
pfile>>balance;
pfile>>address;
pfile>>dat;
pfile>>deposit;
pfile>>withdraw;
withdraw[0]=32;

cout<<account_no<<name<<balance<<dat<<
deposit<<address<<withdraw<<endl;
pfile.close();
strcat(withdraw,"#");
gotoxy(40,13);
cout<<" ";
cin>>tmp;
itoa(tmp,tmp_with,10);
strcat(withdraw,tmp_with);
balance-=tmp;
    ofstream pfile2(name);
    pfile2<<account_no<<endl;
    pfile2<<name<<endl;
    pfile2<<balance<<endl;
    pfile2<<address<<endl;
    pfile2<<dat<<endl;
    pfile2<<deposit<<endl;
    pfile2<<withdraw;
    pfile2.close();
    cleardevice();
    closegraph();
    process();
}
```

Appendix B

Code/Specification of Case Study

System Skeleton of Legacy Bank System

According to analysis of structure of the C language legacy bank system in appendix A, the system skeleton can be presented as follows. As the whold document is quite large, only parts of them are presented here.

```
Program 1
   Call Program2
   Call Program3
   Call Program4
   Call Program5
   Call Program6
End Program 1
      Program2
         Call Program7
End Program2
      Call Program7
      End Program7
Program3
      Call Program8
End Program3
      Program8
            Call Program12
      End Program8
            Program12
            End Program12
Program4
      Call Program8
End Program4
Program5
      Call Program9
      Call Program10
      Call Program11
End Program5
      Program9
      End Program9
      Program10
            Call Program13
            Call Program14
      End Program10
                  Program13
```

```
                        Call Program15
                    End Program13
                        Program15
                        End Program15
                    Program14
                    End Program14
            Program11
            End Program11
        Program6
            Call Program11
End Program6
```

Original Program Fragment of Loan Payment Subroutine

This is the loan payment subroutine in the credit card system of the legacy bank system which is shown in appendix A. It is used to calculate the total repayment amount of a single credit card. The original program fragment of loan payment subroutine which written by C language is shown as follows.

```
(1)    float essential;
(2)    float interest;
(3)    float payment;
(4)    float withdraw;
(5)    interests = interest /100/12;
(6)    payment = payment *12;
(7)    withdraw=0;
(8)    x = pow (1* interests, payment);
(9)    monthly = (essential *x* interests ) / (x-1);
(10)    withdraw= withdraw+i;
(11)   if ((0<monthly) && (monthly < MAXLIMIT))
         { payments = monthly;      total = monthly * payment;
          totalinterest = (minthly*payment - essential;}
(12)   else { payment = 0, total = 0, totalinterest = 0;}
(13)   printf ("%d",withdraw);
(14)   write (payment)
```

Slice of Loan Payment Subroutine with Respect to Criterion (14, {payment})

This shows the slice of the loan payment subroutine with respect to criterion (14, {payment}). It is obtained from the Original code by including only those statements that

could affected the value of the variable payment at line 14. As can be seen in the figure, all computations involving variable withdraw have been slided away. The deleted statements have not effected upon the slicing criterion about payment when the program is executed in any initial states.

```
(1)    float essential;
(2)    float interest;
(3)    float payment;
(4)
(5)    interests = interest /100/12;
(6)    payment = payment *12;
(7)
(8)    x = pow (1* interests, payment);
(9)    monthly = (essential *x* interests ) / (x-1);
(10)
(11)  if ((0<monthly) && (monthly < MAXLIMIT))
          { payments = monthly;      total = monthly * payment;
            totalinterest = (minthly*payment - essential;}
(12)  else { payment = 0, total = 0, totalinterest = 0;}
(13)
(14)  write (payment)
```

DTD Representation of Loan Payment Subroutine

The loan payment subroutine is represented by DTD as follows.

```
<!ELEMENT Legacy Bank System (Credit Card System)>
<!ELEMENT Credit Card System (Loan Payment)>
<!ELEMENT Loan Payment (Interests,Payment)>
<!ELEMENT Interests (Interest,Division)>
<!ATTLIST interests = interest/100/12 CDATA #ReQuired>
<!ATTLIST interests, interest CDATA #ReQuired>
<!ATTLIST 100, 12 CDATA #ReQuired>
<!ELEMENT Payment (Payment,Multiplication)>
<!ATTLIST payment = payment *12 CDATA #ReQuired>
<!ATTLIST payment CDATA #ReQuired>
<!ATTLIST 12 CDATA #ReQuired>
```

Wrapping Loan Payment Subroutine with JNI

This shows the example of wrapping credit card loan payment subroutine in the legacy bank system.. The reusable legacy code in C language (Loan_payments function) is wrapped by JNI technique to provide a Java interface.

```
JNIEXPORT jfloatArray JNICALL
Java_LoanImp_Loan_Payments (JNIEnv *env, jobject obj, jfloatArray arr)
{
     jsize len = ( *env)->GetArrayLength(env.arr);
     jfloatArray sum;
     jfloat *jarry =    ( *env)->GetFloatArrayElements(env,arr,0);

/*float Loan_payments (float essential, float interests, float payment )(
original function call - (no longer necessary)*/
   essential = body [0];
   interests =body[1]; /*Data stream IN*/
   payment = body [2];

     interests = inrerest /100/12;
     payment = payment *12;
     x = pow (1* interests, payment);
     monthly = (essential *x* interests ) / (x-1);
     if ((0<monyhly) && (monthly < MAXLIMIT)) {
          payments = monthly;
          total = monthly * payment;
          totalinterest = (minthly * payment) - essential; }
     else { payment = 0, total = 0, totalinterest = 0; }

     body [0] = payments;
     body [1] = total;
     body [2] =totalinterests;

     jarry [0] = payments;
     jarry [1] = total;
     jarry [2] =totalinterests;
     sum = (*env)-> NewfloatArray (env, len*sizeof(float)+1);
     (*env)->SetFloatArrayRegion (env, sum, 0, 10 * sizeof (float),
     (jfloat *) jarray);
     (*env)-> ReleaseFloatArrayElements(env, arr, jarray, 0);
     return sum; /*Data Stream OUT*/
}
```

XML Component Represented Application

This shows credit card loan payment's XML component represented application. It consists of the <network> information. The network XML data represents a composition of component instance "source" and "payment" together with any users.

```
<application>
    <network>
     <instance componentName="Source"
          componentPackage="Loan_Payments" id="1">
               <property name="degrees of freedom" value="100"/> </instance>
     <instance componentName="Payment"
          componentPackage="Payment" id="2"/>
     <dataflow sinkComponent="2" sinkPort=" interests"
          sourceComponent="1" sourcePort="interests"/>
        ...
    </network>
</application>
```

XML Component Represented Repository

This shows credit card loan payment's XML component represented application. It consists of the <repository> information. The repository XML data provides the interface information for the <component> types, specifying <port> and <property> elements.

```
<repository>
<component package="bank.Loan_payments" name="Source">
<propertyDefinition type=" " name=" " value=" "/>
<port objectPackage="bank.payments" objectName="payment" portName="payment"/>
<implementation language="C" platform="C" url="file:.">
<action portName="payments">
<binding method="getpayments"> ... </binding>
<classPerformanceModel type="initial" url="http:" />
</action>
</implementation>
<implementation language="C" platform="C" url="file:."> ... </implementation>
</component>

<object package="bank.Loan_payments" name="payment" >
<method name="getpayments" type="action">
<argument typeName="payments" typePackage="bank.Loan_payments" />
</method>
</object>
</repository>
```

WSDL Document of CalculatorService

This shows the WSDL code of the CalculatorService. This Calculator.wsdl file is the XML document that describes the calculator service interface. The service interface describes how the outside world can interact with this service, specifically the operations that can be performed on it.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CalculatorService"
targetNamespace=http://examples.strl.org/calculator/CalculatorService

xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://examples.strl.org/calculator/CalculatorService"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wslw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsd
l"
xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    location="../../wsrf/properties/WS-ResourceProperties.wsdl" />

<!== Types == >

<types>
<xsd:schema targetNamespace="http://examples.strl.org/calculator/CalculatorService"
xmlns:tns="http://examples.strl.org/calculator/CalculatorService"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:import
        namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
        schemaLocation="../../ws/addressing/WS-Addressing.xsd" />

<!== Requests and Responses == >

    <xsd:element name="add" type="xsd:int"/>
    <xsd:element name="addResponse">
        <xsd:complexType/>
    </xsd:element>

    <xsd:element name="subtract" type="xsd:int"/>
    <xsd:element name="subtractResponse">
        <xsd:complexType/>
    </xsd:element>

    <xsd:element name="getValueRP">
```

```
            <xsd:complexType/>
        </xsd:element>
        <xsd:element name="getValueRPResponse" type="xsd:int"/>

<!== Resources Properties == >

        <xsd:element name="LegacyBankSystem" type="xsd:string"/>
        <xsd:element name="CreditCardSystem" type="xsd:string"/>
        <xsd:element name="LoanPayment" type="xsd:string"/>
        <xsd:element name="Interests" type="xsd:string"/>
        <xsd:element name="Payment" type="xsd:string"/>
        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>
.....

        <xsd:element name="CalculatorResourceProperties">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
                    <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>

<!== Messages == >

<message name="AddInputMessage">
<part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
<part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
<part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
<part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="GetValueRPInputMessage">
<part name="parameters" element="tns:getValueRP"/>
</message>
<message name="GetValueRPOutputMessage">
<part name="parameters" element="tns:getValueRPResponse"/>
</message>

<!== Porttype == >

<portType name="CalculatorPortType"
wsdlpp:extends="wsrpw:GetResourceProperty"
wsrp:ResourceProperties="tns:CalculatorResourceProperties">

<operation name="add">
<input message="tns:AddInputMessage"/>
```

```
<output message="tns:AddOutputMessage"/>
</operation>

<operation name="subtract">
<input message="tns:SubtractInputMessage"/>
<output message="tns:SubtractOutputMessage"/>
</operation>

<operation name="getValueRP">
<input message="tns:GetValueRPInputMessage"/>
<output message="tns:GetValueRPOutputMessage"/>
</operation>

</portType>
</definitions>
```

QNames.java Document of CalculatorService

The CalculatorQNames.java file is a convenient interface class containing the QName URI/namespace constants relevant to the proposed Grid service. By having the service classes implement this interface, these constants can be referred to replicate themselves throughout the project. This is written by Java language.

```
package org.strl.examples.services.Calculator.impl;
import javax.xml.namespace.QName;
public interface CalculatorQNames
{
      public static final String NS = "http://examples.strl.org/calculator/calculator service";
      public static final QName RP_VALUE = new QName(NS, "Value");
      public static final QName RP_LASTOP = new QName(NS, "LastOp");
      public static final QName RESOURCE_PROPERTIES = new
      QName(NS,"calculatorResourceProperties");
}
```

Request Message Representation

This show the request message of the example. It represents a request message used to retrieve two resource property elements from the WS-Resource that implements the loan payment portType. This identifier information is carried in the SOAP header element.

```
<soap:Envelope>
      <soap:Header>
      <tns:resourceID> Loan Payment </tns:resourceID>
      </soap:Header>
      <soap:Body>
            <wsrp:GetMultipleResourceProperty>
                xmlns:tns=     https://... >
                <wsrp:ResourceProperty>
                    tns: Interests
                </wsrp:ResourceProperty>
                <wsrp:ResourceProperty>
                    tns: Payment
                </wsrp:ResourceProperty>
           </wsrp:GetMultipleResourceProperty>
      </soap:Body>
</soap:Envelope>
```

Response Message Representation

This show the response message of the example. It represents the result of response value of the request. This identifier information is carried in the SOAP header element.

```
<soap:Envelope>
    <soap:Body>
        <wsrp:GetMultipleResourcePropertyResponse>
        <Interests> “interests”</Interests>
        </wsrp:GetMultipleResourcePropertyResponse>
        <wsrp:GetMultipleResourcePropertyResponse>
        <Payment> “payment”</Payment>
        </wsrp:GetMultipleResourcePropertyResponse>
    </soap:Body>
</soap:Envelope>
```

Service Implementation Document of CalculatorService

This shows the service implementation document of CalculatorService. The CalculatorService.java file is the service implementation that provides the core

functionality for exposing local directory information. The source for this Java class is shown as follows.

```
package org.strl.examples.services.Calculator.impl;
import java.rmi.RemoteException;

import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;

import org.strl.examples.Calculator.CalculatorService.AddResponse;
import org.strl.examples.Calculator.CalculatorService.SubtractResponse;
import org.strl.examples.Calculator.CalculatorService.GetValueRP;

public class CalculatorService implements Resource, ResourceProperties
{
    /* Resource Property set */
    private ResourcePropertySet propSet;

    /* Resource properties */
    private int value;
    private String lastOp;

    /* Constructor. Initialises RPs */
    public CalculatorService() throws RemoteException {
        /* Create RP set */
        this.propSet = new
SimpleResourcePropertySet(CalculatorQNames.RESOURCE_PROPERTIES);

        /* Initialise the RPs */
        try {
            ResourceProperty valueRP = new
        ReflectionResourceProperty(CalculatorQNames.RP_VALUE, "Value", this);
            this.propSet.add(valueRP);
            setValue(0);
            ResourceProperty lastOpRP = new ReflectionResourceProperty(
            CalculatorQNames.RP_LASTOP, "LastOp", this);
            this.propSet.add(lastOpRP);
            setLastOp("NONE");
        } catch (Exception e) {
                throw new RuntimeException(e.getMessage());
        }
    }

    /* Get/Setters for the RPs */
    public int getValue() {
        return value;
    }
    public void setValue(int value) {
        this.value = value;
```

```
}
public String getLastOp() {
        return lastOp;
}
public void setLastOp(String lastOp) {
        this.lastOp = lastOp;
}

/* Remotely-accessible operations */
public AddResponse add(int a) throws RemoteException {
        value += a;
        lastOp = "ADDITION";
        return new AddResponse();
}

public SubtractResponse subtract(int a) throws RemoteException {
        value -= a;
        lastOp = "SUBTRACTION";
        return new SubtractResponse();
}
public int getValueRP(GetValueRP params) throws RemoteException {
return value;
}

/* Required by interface ResourceProperties */
public ResourcePropertySet getResourcePropertySet() {
        return this.propSet;
}
}
```

JNDI Deployment Document of CalculatorService

This shows the JNDI deployment document of CalculatorService. The deploy-jndi-config Xml file is the JNDI deploy file that enables the GT4 WSRF implementation to locate the resource home for this service.

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

<service name="examples/core/first/CalculatorService">
    <resource name="home" type="org.globus.wsrf.impl.ServiceResourceHome">
    <resourceParams>

        <parameter>
            <name>factory</name>
            <value>org.globus.wsrf.jndi.BeanFactory</value>
        </parameter>
```

```
        </resourceParams>

        </resource>
    </service>
    </jndiConfig>
```

WSDD Deployment Document of CalculatorService

This shows the WSDD deployment document of CalculatorService. The deploy-server.wsdd XML file is the configuration file that tells the services container how to publish the service.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="examples/CalculatorService" provider="Handler" use="literal"
    style="document">
        <parameter name="className"
    value="org.globus.examples.services.Calculator.impl.CalculatorService"/>

        <wsdlFile>share/schema/examples/CalculatorService/CalculatorService.wsdl</wsdlFile
    >
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>
</deployment>
```

Buildservice.xml Document of CalculatorService

This shows the Buildservice.xml Document of CalculatorService. It is a general-purpose Ant buildfile and script that, given a set of Java, WSDL, WSDD, etc. files conforming to a specific directory structure, and will generate a GAR file without the need to manually edit the Ant file.

```
<?xml version="1.0"?>
<project default="all" name="Grid Service Buildfile" basedir=".">
    <description>
        Grid Service Buildfile
    </description>
    <property environment="env"/>
    <target name="all">
        <ant antfile="${build.gar}" target="clean"/>
        <ant antfile="${build.gar}" target="all"/>
        <ant antfile="${build.packages}" target="deployGar"/>
        <ant antfile="${build.tomcat}" target="deploySecureTomcat"
        dir="/Dev/GTK/share/globus_wsrf_common/tomcat" />
    </target>
</project>
```

Buildservice.properties Document of CalculatorService

This shows the Buildservice.properties Document of CalculatorService. The buildservice.properties file contains the "name=value properties specific to this service and need to guide the various build tasks through the GAR creation and deployment process.

```
package=com.strl.examples.services.calculator
interface.name=Calculator
package.dir=org/strl/examples/services/Calculator
schema.path=examples/CalculatorService
service.name=CalculatorService
gar.filename=org_strl_examples_calculator

build.gar=/dev/GTK/etc/build.xml
build.packages=/Dev/GTK/share/globus_wsrf_common/build-packages.xml
build.tomcat=/Dev/GTK/share/globus_wsrf_common/tomcat/tomcat.xml
gar.name=/Dev/eclipse/workspace/ProvisionDirService/org_strl_examples_calculator
tomcat.dir=/Dev/tomcat5
```

Appendix C

List of Publications by Author

[1] Jianzhi Li and Hongji Yang, "Reengineering Websites into Stateful Resources for Grid Service Oriented Evolution", *International Journal of Multiagent and Grid Systems*, IOS Press, Vol. 2, No. 2, 2006.

[2] Jianzhi Li, Zhuopeng Zhang, Bing Qiao and Hongji Yang, "A Component Mining Approach to Incubate Grid Services in Object-Oriented Legacy Systems", *International Journal of Automation and Computing*, Vol. 3, No. 1, 2006.

[3] Jianzhi Li and Hongji Yang, "Incorporating Legacy System into Semantic Grid Framework", *IEEE International Conference on Information Reuse and Integration (IRI)*, Hawaii, USA, 2006.

[4] Jianzhi Li and Hongji Yang, "Developing Grid Middleware for Software Evolution", *(Position Paper) IEEE International Software Technology and Engineering Practice Conference (STEP)*, Budapest, Hungary, 2005.

[5] Jianzhi Li and Hongji Yang, "Towards Evolving Web Sites into Grid Services Environment", *IEEE International Symposium on Web Site Evolution (WSE)*, Budapest, Hungary, 2005.

[6] Jianzhi Li, Zhuopeng Zhang and Hongji Yang, "A Grid Oriented Approach to Reusing Legacy Code in ICENI Framework", *IEEE International Conference on Information Reuse and Integration (IRI)*, Las Vegas, USA, 2005.

[7] Jian Kang, Jianzhi Li, Shaoyun Li, Hongji Yang, "Band Reservation System Design and Analysis for Grid Service", *European Young Researchers Workshop on Service Oriented Computing (YR-WSOC)*, Leicester, UK, 2005.

[8] Jianzhi Li and Hongji Yang, "Leveraging Legacy Assets for Grid Applications", *Postgraduate Research Conference in Electronics, Photonics, Communications and Networks, and Computing Science (PREP)*, Lancaster, UK, 2005.

[9] Jianzhi Li and Hongji Yang, "Evolving Distributed Legacy Systems into Dynamic Grid Services", *International Workshop on Automation, Computing and Manufacturing*, Beijing, China, 2005.

[10] Jianzhi Li and Hongji Yang, "An Approach to Reengineering Legacy System into Grid Environment", *10th Annual Conference of Chinese Automation and Computing Society in the UK (CACSUK)*, Liverpool, UK, 2004.